15

20

25

30



Furthermore, while the DRAM interface is compression standard-independent, it still must be configured to implement each of the multiple standards, H.261, JPEG and MPEG. How the DRAM interface is reconfigured for multistandard operation will be subsequently further described herein.

Accordingly, to understand the operation of the DRAM interface requires an understanding of the relationship between the DRAM interface and the address generator, and how the two communicate using the two wire interface.

In general, as its name implies, the address generator generates the addresses the DRAM interface needs in order to address the DRAM (e.g., to read from or to write to a particular address in DRAM). With a two-wire interface, reading and writing only occurs when the DRAM interface has both data (from preceding stages in the pipeline), and a valid address (from address generator). The use of a separate address generator simplifies the construction of both the address generator and the DRAM interface, as discussed further below.

In the present invention, the DRAM interface can operate from a clock which is asynchronous to both the address generator and to the clocks of the stages through which data is passed. Special techniques have been used to handle this asynchronous nature of the operation.

Data is typically transferred between the DRAM interface and the rest of the chip in blocks of 64 bytes (the only exception being prediction data in the Temporal Decoder). Transfers take place by means of a device known as a "swing buffer". This is essentially a pair of RAMs operated in a double-buffered configuration, with the DRAM interface filling or emptying one RAM while another part of the chip empties or fills the other RAM. A separate bus which carries an address from an address generator is associated with each

15

20

25

30

swing buffer.

In the present invention, each of the chips has four swing buffers, but the function of these swing buffers is different In the spatial decoder, one swing buffer is in each case. used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, however, one swing buffer is used to write intra or predicted picture data to the DRAM, the second to read intra or predicted data from the DRAM and the other two are used to read forward and backward prediction data. In the video formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data from the DRAM, one for each of luminance (Y) and the red and blue color difference data (Cr and Cb, respectively).

The following section describes the operation of a hypothetical DRAM interface which has one write swing buffer and one read swing buffer. Essentially, this is the same as the operation of the Spatial Decoder's DRAM interface. The operation is illustrated in Figure 23.

Figure 23 illustrates that the control interfaces between the address generator 301, the DRAM interface 302, and the remaining stages of the chip which pass data are all The address generator 301 may either two wire interfaces. generate addresses as the result of receiving control tokens, or it may merely generate a fixed sequence of addresses (e.g., for the FIFO buffers of the Spatial Decoder). DRAM interface treats the two wire interfaces associated with the address generator 301 in a special way. keeping the accept line high when it is ready to receive an address, it waits for the address generator to supply a valid address, processes that address and then sets the accept line implements a it high for one clock period. Thus,

10

15

20

25

30

request/acknowledge (REQ/ACK) protocol.

A unique feature of the DRAM interface 302 is its ability to communicate independently with the address generator 301 and with the stages that provide or accept the For example, the address generator may generate an address associated with the data in the write swing buffer (Figure 24), but no action will be taken until the Write swing buffer signals that there is a block of data ready to be written to the external DRAM. Similarly, the write swing buffer may contain a block of data which is ready to be written to the external DRAM, but no action is taken until an address is supplied on the appropriate bus from the address generator 301. Further, once one of the RAMs in the write swing buffer has been filled with data, the other may be completely filled and "swung" to the DRAM interface side before the data input is stalled (the two-wire interface accept signal set low).

In understanding the operation of the DRAM interface 302 of the present invention, it is important to note that in a properly configured system, the DRAM interface will be able to transfer data between the swing buffers and the external DRAM 303 at least as fast as the sum of all the average data rates between the swing buffers and the rest of the chip.

Each DRAM interface 302 determines which swing buffer it will service next. In general, this will either be a "round robin" (i.e., the next serviced swing buffer is the next available swing buffer which has least recently had a turn), or a priority encoder, (i.e., in which some swing buffers have a higher priority than others). In both cases, an additional request will come from a refresh request generator which has a higher priority than all the other requests. The refresh request is generated from a refresh counter which can be programmed via the microprocessor interface.

Referring now to Figure 24, there is shown a block

NI I

10

15

20

25

30

diagram of a write swing buffer. The write swing buffer interface includes two blocks of RAM, RAM1 311 and RAM2 312. As discussed further herein, data is written into RAM1 311 and RAM2 312 from the previous stage, under the control of the write address 313 and control 314. From RAM1 311 and RAM2 312, the data is written into DRAM 515. When writing data into DRAM 315, the DRAM row address is provided by the address generator, and the column address is provided by the write address and control, as described further herein. operation, valid data is presented at the input 316 (data Typically, the data is received from the previous As each piece of data is accepted by the DRAM interface, it is written into RAM1 311 and the write address control increments the RAM1 address to allow the next piece of data to be written into RAM1. Data continues to be written into RAM1 311 until either there is no more data, or RAM1 is full. When RAM1 311 is full, the input side gives up control and sends a signal to the read side to indicate that RAM1 is now ready to be read. This signal passes between two asynchronous clock regimes and, therefore, passes through three synchronizing flip flops.

Provided RAM2 312 is empty, the next item of data to arrive on the input side is written into RAM2. Otherwise, this occurs when RAM2 312 has emptied. When the round robin or priority encoder (depending on which is used by the particular chip) indicates that it is now the turn of this swing buffer to be read, the DRAM interface reads the contents of RAM1 311 and writes them to the external DRAM 315. A signal is then sent back across the asynchronous interface, to indicate that RAM1 311 is now ready to be filled again.

If the DRAM interface empties RAM1 311 and "swings" it before the input side has filled RAM2 312 , then data can be

10

15

2.0

25

30

accepted by the swing buffer continually. Otherwise, when RAM2 is filled, the swing buffer will set its accept single low until RAM1 has been "swung" back for use by the input side.

The operation of a read swing buffer, in accordance with the present invention, is similar, but with the input and output data busses reversed.

The DRAM interface of the present invention is designed to maximize the available memory bandwidth. Each 8x8 block of data is stored in the same DRAM page. In this way, full use can be made of DRAM fast page access modes, where one row address is supplied followed by many column addresses. In particular, row addresses are supplied by the address generator, while column addresses are supplied by the DRAM interface, as discussed further below.

In addition, the facility is provided to allow the data bus to the external DRAM to be 8, 16 or 32 bits wide. Accordingly, the amount of DRAM used can be matched to the size and bandwidth requirements of the particular application.

In this example (which is exactly how the DRAM interface on the Spatial Decoder works) the address generator provides the DRAM interface with block addresses for each of the read and write swing buffers. This address is used as the row address for the DRAM. The six bits of column address are supplied by the DRAM interface itself, and these bits are also used as the address for the swing buffer RAM. The data bus to the swing buffers is 32 bits wide. Hence, if the bus width to the external DRAM is less than 32 bits, two or four external DRAM accesses must be made before the next word is read from a write swing buffer or the next word is written to a read swing buffer (read and write refer to the direction of transfer relative to the external DRAM).

The situation is more complex in the case of the

10

15

20

25

30

Temporal Decoder and the Video Formatter. The Temporal Decoder's addressing is more complex because of its predictive aspects as discussed further in this section. The video formatter's addressing is more complex because of multiple video output standard aspects, as discussed further in the sections relating to the video formatter.

As mentioned previously, the Temporal Decoder has four swing buffers: two are used to read and write decoded intra and predicted (I and P) picture data. These operate as described above. The other two are used to receive prediction data. These buffers are more interesting.

In general, prediction data will be offset from the position of the block being processed as specified in the motion vectors in x and y. Thus, the block of data to be retrieved will not generally correspond to the block boundaries of the data as it was encoded (and written into the DRAM). This is illustrated in Figure 25, where the shaded area represents the block that is being formed whereas the dotted outline represents the block from which it is being predicted. The address generator converts the address specified by the motion vectors to a block offset (a whole number of blocks), as shown by the big arrow, and a pixel offset, as shown by the little arrow.

In the address generator, the frame pointer, base block address and vector offset are added to form the address of the block to be retrieved from the DRAM. If the pixel offset is zero, only one request is generated. If there is an offset in either the x or y dimension then two requests are generated, i.e., the original block address and the one immediately below. With an offset in both x and y, four requests are generated. For each block which is to be retrieved, the address generator calculates start and stop addresses which is best illustrated by an example.

Consider a pixel offset of (1,1), as illustrated by the

10

15

20

25

30

shaded area in Figure 26. The address generator makes four requests, labelled A through D in the Figure. The problem to be solved is how to provide the required sequence of row addresses quickly. The solution is to use "start/stop" technology, and this is described below.

Consider block A in Figure 26. Reading must start at position (1,1) and end at position (7,7). Assume for the moment that one byte is being read at a time (i.e., an 8 bit DRAM interface). The x value in the co-ordinate pair forms the three LSBs of the address, the y value the three MSB. The x and y start values are both 1, providing the address, 9. Data is read from this address and the x value is The process is repeated until the x value incremented. reaches its stop value, at which point, the y value is incremented by 1 and the x start value is reloaded, giving an address of 17. As each byte of data is read, the x value is again incremented until it reaches its stop value. process is repeated until both x and y values have reached their stop values. Thus, the address sequence of 9, 10, 11, 12, 13, 14, 15, 17..., 23, 25, ..., 31, 33,..., 57,...,63 is generated.

In a similar manner, the start and stop co-ordinates for block B are: (1,0) and (7,0), for block C: (0,1) and (0,7), and for block D: (0,0) and (0,0).

The next issue is where this data should be written. Clearly, looking at block A, the data read from address 9 should be written to address 0 in the swing buffer, while the data from address 10 should be written to address 1 in the swing buffer, and so on. Similarly, the data read from address 8 in block B should be written to address 15 in the swing buffer and the data from address 16 should be written to address 15 in the swing buffer. This function turns out to have a very simple implementation, as outlined below.

Consider block A. At the start of reading, the swing

15

20

25

30

buffer address register is loaded with the inverse of the stop value. The y inverse stop value forms the 3 MSBs and the x inverse stop value forms the 3 LSB. In this case, while the DRAM interface is reading address 9 in the external DRAM, the swing buffer address is zero. The swing buffer address register is then incremented as the external DRAM address register is incremented, as consistent with proper prediction addressing.

The discussion so far has centered on an 8 bit DRAM In the case of a 16 or 32 bit interface, a few minor modifications must be made. First, the pixel offset vector must be "clipped" so that it points to a 16 or 32 bit boundary. In the example we have been using, for block A, the first DRAM read will point to address 0, and data in Second, the unwanted addresses 0 through 3 will be read. data must be discarded. This is performed by writing all the data into the swing buffer (which must now be physically larger than was necessary in the 8 bit case) and reading with an offset. When performing MPEG half-pel interpolation, 9 bytes in x and/or y must be read from the DRAM interface. this case, the address generator provides the appropriate start and stop addresses. Some additional logic in the DRAM interface is used, but there is no fundamental change in the way the DRAM interface operates.

The final point to note about the Temporal Decoder DRAM interface of the present invention, is that additional information must be provided to the prediction filters to indicate what processing is required on the data. This consists of the following:

a "last byte" signal indicating the last byte of a transfer (of 64,72 or 81 bytes);

an H.261 flag;

a bidirectional prediction flag;

two bits to indicate the block's dimensions (8 or 9 bytes

in \times and y); and

10

15

20

25

30

a two bit number to indicate the order of the blocks.

The last byte flag can be generated as the data is read out of the swing buffer. The other signals are derived from the address generator and are piped through the DRAM interface so that they are associated with the correct block of data as it is read out of the swing buffer by the prediction filter block.

In the Video Formatter, data is written into the external DRAM in blocks, but is read out in raster order. Writing is exactly the same as already described for the Spatial Decoder, but reading is a little more complex.

The data in the Video Formatter, external DRAM is organized so that at least 8 blocks of data fit into a single page. These 8 blocks are 8 consecutive horizontal blocks. When rasterizing, 8 bytes need to be read out of each of 8 consecutive blocks and written into the swing buffer (1.e., the same row in each of the 8 blocks).

Considering the top row (and assuming a byte-wide interface), the x address (the three LSBS) is set to zero, as is the y address (3 MSBS). The x address is then incremented as each of the first 8 bytes are read out. At this point, the top part of the address (bit 6 and above - LSB = bit 0) is incremented and the x address (3 LSBS) is reset to zero. This process is repeated until 64 bytes have been read. With a 16 or 32 bit wide interface to the external DRAM the x address is merely incremented by two or four, respectively, instead of by one.

In the present invention, the address generator can signal to the DRAM interface that less than 64 bytes should be read (this may be required at the beginning or end of a raster line), although a multiple of 8 bytes is always read. This is achieved by using start and stop values. The start value is used for the top part of the address (bit 6 and

enter the second of the second

15

20

25

30

above), and the stop value is compared with the start value to generate the signal which indicates when reading should stop.

The DRAM interface timing block in the present invention uses timing chains to place the edges of the DRAM signals to a precision of a quarter of the system clock period. Two quadrature clocks from the phase locked loop are used. These are combined to form a notional 2x clock. Any one chain is then made from two shift registers in parallel, on opposite phases of the 2x clock.

First of all, there is one chain for the page start cycle and another for the read/write/refresh cycles. The length of each cycle is programmable via the microprocessor interface, after which the page start chain has a fixed length, and the cycle chain's length changes as appropriate during a page start.

On reset, the chains are cleared and a pulse is created. The pulse travels along the chains and is directed by the state information from the DRAM interface. The pulse generates the DRAM interface clock. Each DRAM interface clock period corresponds to one cycle of the DRAM, consequently, as the DRAM cycles have different lengths, the DRAM interface clock is not at a constant rate.

Moreover, additional timing chains combine the pulse from the above chains with the information from the DRAM interface to generate the output strobes and enables such as notcas, notras, notwe, notbe.

12. PREDICTION FILTERS

Referring again to Figures 12, 17, 18, and more particularly to Figure 12, there is shown a block diagram of the Temporal Decoder. This includes the prediction filter. The relationship between the prediction filter and the rest of the elements of the temporal decoder is shown in greater

princip (

1 1 1 1 1 1

10

15

20

25

30

detail in Figure 17. The essence of the structure of the prediction filter is shown in Figures 18 and 28. A detailed description of the operation of the prediction filter can be found in the section, "More Detailed Description of the Invention."

In general, the prediction filter in accordance with the present invention, is used in the MPEG and H.261 modes, but not in the JPEG mode. Recall that in the JPEG mode, the Temporal Decoder just passes the data through to the Video Formatter, without performing any substantive decoding beyond that accomplished by the Spatial Decoder. Referring again to Figure 18, in the MPEG mode the forward and backward prediction filters are identical and they filter the respective MPEG forward and backward prediction blocks. In the H.261 mode, however, only the forward prediction filter is used, since H.261 does not use backward prediction.

Each of the two prediction filters of the present invention is substantially the same. Referring again to Figures 18 and 28 and more particularly to Figure 28, there is shown a block diagram of the structure of a prediction filter. Each prediction filter consists of four stages in series. Data enters the format stage 331 and is placed in a format that can be readily filtered. In the next stage 332 an I-D prediction is performed on the X-coordinate. the necessary transposition is performed by a dimension buffer stage 333, an I-D prediction is performed on the Ycoordinate in stage 334. How the stage perform the filtering is further described in greater detail subsequently. Which filtering operations are required, are defined by the compression standard. In the case of H.261, the actual filtering performed is similar to that of a low pass filter.

Referring again to Figure 17, multi-standard operation requires that the prediction filters be reconfigurable to perform either MPEG or H.261 filtering, or

11 to pg 1 1#

process a H

15

20

25

30

to perform no filtering at all in JPEG mode. As with many other reconfigurable aspects of the three chip system, the prediction filter is reconfigured by means of tokens. Tokens are also used to inform the address generator of the particular mode of operation. In this way, the address generator can supply the prediction filter with the addresses of the needed data, which varies significantly between MPEG and JPEG.

13. ACCESSING REGISTERS

Most registers in the microprocessor interface (MPI) can only be modified if the stage with which they are associated is stopped. Accordingly, groups of registers will typically be associated with an access register. The value zero in an access register indicates that the group of registers associated with that particular access register should not be modified. Writing 1 to an access register requests that a stage be stopped. The stage may not stop immediately, however, so the stages access register will hold the value, zero, until it is stopped.

Any user software associated with the MPI and used to perform functions by way of the MPI should wait "after writing a 1 to a request access register" until 1 is read from the access register. If a user writes a value to a configuration register while its access register is set to zero, the results are undefined.

14. MICRO-PROCESSOR INTERFACE

A standard byte wide micro-processor interface (MPI) is used on all circuits with in the Spatial Decoder and Temporal Decoder. The MPI operates asynchronously with various Spatial and Temporal Decoder clocks. Referring to Table A.6.1 of the subsequent further detailed description, there is shown the various MPI signals that

15

20

25

30

are used on this interface. The character of the signal is shown on the input/output column, the signal name is shown on the signal name column and a description of the function of the signal is shown in the description column. The MPI electrical specification are shown with reference to Table A.6.2. All the specifications are classified according to type and there types are shown in the column entitled symbol. The description of what these symbols represent is shown in the parameter column. The actual specifications are shown in the respective columns min, max and units.

The DC operating conditions can be seen with reference to Table A.6.3. Here the column headings are the same as with reference to Table A.6.2. The DC electrical characteristics are shown with reference to Table A.6.4 and carry the same column headings as depicted in Tables A.6.2 and A.6.3.

15. MPI READ TIMING

The AC characteristics of the MPI read timing diagrams are shown with reference to Figure 54. Each line of the Figure is labelled with a corresponding signal name and the timing is given in nano-seconds. The full microprocessor interface read timing characteristics are shown with reference to Table A.6.5. The column entitled Number is used to indicate the signal corresponding to the name of that signal as set forth in the characteristic column. The columns identified by MIN and MAX provide the minimum length of time that the signal is present the maximum amount of time that this signal is available. The Units column gives the units of measurement used to describe the signals.

16. MPI WRITE TIMING

The general description of the MPI write timing diagrams

10

15

20

are shown with reference to Figure 54. This Figure shows each individual signal name as associated with the MPI write timing. The name, the characteristic of the signal, and other various physical characteristics are shown with reference to Table 6.6.

17. KEYHOLE ADDRESS LOCATIONS

In the present invention, certain less frequently accessed memory map locations have been placed behind keyhole registers. A keyhole register has two registers associated with it. The first register is a keyhole address register and the second register is a keyhole data register. The keyhole address specifies a location within a extended address space. A read or a write operation to a keyhole data register accesses the locations specified by the keyhole address register. After accessing a keyhole data register, the associated keyhole address register increments. Random access within the extended address space is only possible by writing in a new value to the keyhole address register for each access. A circuit within the present invention may have more than one keyhole memory maps. Nonetheless, there is no interaction between the different keyholes.

18. PICTURE-END

Referring again to Figure 11, there is shown a

general block diagram of the Spatial Decoder used in the
present invention. It is through the use of this block
diagram that the function of PICTURE_END will be described.
The PICTURE_END function has the multi-standard advantage
of being able to handle H.261 encoded picture information,

MPEG and JPEG signals.

As previously described, the system of Figure 11 is interconnected by the two wire interface previously

15

20

25

30

described. Each of the functional blocks is arranged to operate according to the state machine configuration shown with reference to Figure 10.

In general, the PICTURE_END function in accordance with the invention begins at the Start Code Detector which generates a PICTURE_END control token. The PICTURE_END control token is passed unaltered through the start-up control circuit to the DRAM interface. Here it is used to flush out the write swing buffers in the DRAM interface.

Recall, that the contents of a swing buffer are only written to RAM when the buffer is full. However, a picture may end at a point where the buffer is not full, therefore, causing the picture data to become stuck. The PICTURE_END token forces the data out of the swing buffer.

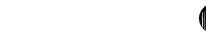
Since the present invention is a multi-standard machine, the machine operates differently for each compression standard. More particularly, the machine is fully described as operating pursuant to machine-dependent action cycles. For each compression standard, a certain number of the total available action cycles can be selected by a combination of control tokens and/or output signals from the MPU or they can be selected by the design of the control tokens themselves. In this regard, the present invention is organized so as to delay the information from going into subsequent blocks until all of the information has been collected in an upstream block. The system waits until the data has been prepared for passing to the next stage. In this way, the PICTURE END signal is applied to the coded data buffer, and the control portion of the PICTURE END signal causes the contents of the data buffers to be read and applied to the Huffman decoder and video demultiplexor circuit.

Another advantage of the PICTURE_END control token is to identify, for the use by the Huffman decoder

15

2.0

25



demultiplexor, the end of picture even though it has not had the typically expected full range and/or number of signals applied to the Huffman decoder and video demultiplexor circuit. In this situation, the information held in the coded data buffer is applied to the Huffman decoder and video demultiplexor as a total picture. In this way, the state machine of the Huffman decoder and video demultiplexor can still handle the data according to system design.

Another advantage of the PICTURE_END control token is its ability to completely empty the coded data buffer so that no stray information will inadvertently remain in the off chip DRAM or in the swing buffers.

Yet another advantage of the PICTURE_END function is its use in error recovery. For example, assume the amount of data being held in the coded data buffer is less than is typically used for describing the spatial information with reference to a single picture. Accordingly, the last picture will be held in the data buffer until a full swing buffer, but, by definition, the buffer will never fill. At some point, the machine will determine that an error condition exits. Hence, to the extent that a PICTURE_END token is decoded and forces the data in the coded data buffers to be applied to the Huffman decoder and video demultiplexor, the final picture can be decoded and the information emptied from the buffers. Consequently, the machine will not go into error recovery mode and will successfully continue to process the coded data.

A still further advantage of the use of a PICTURE_END token is that the serial pipeline processor will continue the processing of uninterrupted data. Through the use of a PICTURE_END token, the serial pipeline processor is configured to handle less than the expected amount of data and, therefore, continues processing. Typically, a prior

15

20

25

30

art machine would stop itself because of an error condition. As previously described, the coded data buffer counts macroblocks as they come into its storage area. addition, the Huffman Decoder and Video Demultiplexor generally know the amount of information expected for decoding each picture, i.e., the state machine portion of the Huffman decode and Video Demultiplexor know the number of blocks that it will process during each picture recovery cycle. When the correct number of blocks do not arrive from the coded data buffer, typically an error recovery routine would result. However, with the PICTURE_END control token having reconfigured the Huffman Decoder and Video Demultiplexor, it can continue to function because the reconfiguration tells the Huffman Decoder and Video Demultiplexor that it is, indeed, handling the proper amount of information.

Referring again to Figure 10, the Token Decoder portion of the Buffer Manager detects the PICTURE_END control token generated by the Start Code Detector. Under normal operations, the buffer registers fill up and are emptied, as previously described with reference to the normal operation of the swing buffers. Again, a swing buffer which is partially full of data will not empty until it is totally filled and/or it knows that it is time to empty. The PICTURE_END control token is decoded in the Token Decoder portion of the Buffer Manager, and it forces the partially full swing buffer to empty itself into the coded data buffer. This is ultimately passed to the Huffman Decoder and Video Demultiplexor either directly or through the DRAM interface.

19. FLUSHING OPERATION

Another advantage of the PICTURE_END control token is its function in connection with a FLUSH token. The FLUSH

10

20

25

token is not associated with either controlling the reconfiguration of the state machine or in providing data for the system. Rather, it completes prior partial signals for handling by the machine-dependent state machines. Each of the state machines recognizes a FLUSH control token as information not to be processed. Accordingly, the FLUSH token is used to fill up all of the remaining empty parts of the coded data buffers and to allow a full set of information to be sent to the Huffman Decoder and Video Demultiplexor. In this way, the FLUSH token is like padding for buffers.

The Token Decoder in the Huffman circuit recognizes the FLUSH token and ignores the pseudo data that the FLUSH token has forced into it. The Huffman Decoder then operates only on the data contents of the last picture buffer as it existed prior to the arrival of the PICTURE END token and FLUSH token. A further advantage of the use of the PICTURE END token alone or in combination with a FLUSH token is the reconfiguration and/or reorganization of the Huffman Decoder circuit. With the arrival of the PICTURE END token, the Huffman Decoder circuit knows that it will have less information than normally expected to decode the last picture. The Huffman decode circuit finishes processing the information contained in the last picture, and outputs this information through the DRAM interface into the Inverse Modeller. Upon the identification of the last picture, the Huffman Decoder goes into its cleanup mode and readjusts for the arrival of the next picture information.

30 20. FLUSH FUNCTION

The FLUSH token, in accordance with the present invention, is used to pass through the entire pipeline processor and to ensure that the buffers are emptied and that other circuits are reconfigured to await the arrival

10

15

20

25

of new data. More specifically, the present invention comprises a combination of a PICTURE_END token, a padding word and a FLUSH token indicating to the serial pipeline processor that the picture processing for the current picture form is completed. Thereafter, the various state machines need reconfiguring to await the arrival of new data for new handling. Note also that the FLUSH Token acts as a special reset for the system. The FLUSH token resets each stage as it passes through, but allows subsequent stages to continue processing. This prevents a loss of data. In other words, the FLUSH token is a variable reset, as opposed to, an absolute reset.

21. STOP-AFTER PICTURE

The STOP AFTER PICTURE function is employed to shut down the processing of the serial pipeline decompressing circuit at a logical point in its operation. At this point, a PICTURE_END token is generated indicating that data is finished coming in from the data input line, and the padding operation has been completed. The padding function fills partially empty DATA tokens. A FLUSH token is then generated which passes through the serial pipeline system and pushes all the information out of the registers and forces the registers back into their neutral stand-by The STOP_AFTER_PICTURE event is then generated and no more input is accepted until either the user or the system clears this state. In other words, while a PICTURE END token signals the end of a picture, the STOP AFTER PICTURE operation signals the end of all current processing.

30 22. MULTI-STANDARD - SEARCH MODE

Another feature of the present invention is the use of a SEARCH MODE control token which is used to reconfigure

10

15

20

25

30

the input to the serial pipeline processor to look at the incoming bit stream. When the search mode is set, the Start Code Detector searches only for a specific start code or marker used in any one of the compression standards. It will be appreciated, however, that, other images from other data bitstreams can be used for this purpose. Accordingly, these images can be used throughout this present invention to change it to another embodiment which is capable of using the combination of control tokens, and DATA tokens along with the reconfiguration circuits, to provide similar processing.

The use of search mode in the present invention is convenient in many situations including 1) if a break in the data bit stream occurs; 2) when the user breaks the data bit stream by purposely changing channels, e.g., data arriving, by a cable carrying compressed digital video; or 3) by user activation of fast forward or reverse from a controllable data source such as an optical disc or video disc. In general, a search mode is convenient when the user interrupts the normal processing of the serial pipeline at a point where the machine does not expect such an interruption.

When any of the search modes are set, the Start Code Detector looks for incoming start images which are suitable for creating the machine independent tokens. All data coming into the Start Code Detector prior to the identification of standard-dependent start images is discarded as meaningless and the machine stands in an idling condition as it waits this information.

The Start Code Detector can assume any one of a number of configurations. For example, one of these configurations allows a search for a group of pictures or higher start codes. This pattern causes the Start Code Detector to discard all its input and look for the

15

25

30

group_start standard image. When such an image is identified, the Start Code Detector generates a GROUP_START token and the search mode is reset automatically.

It is important to note that a single circuit, the Huffman Decoder and Video Demultiplex circuit, is operating with a combination of input signals including the standard-independent set-up signals, as well as, the CODING_STANDARD signals. The CODING_STANDARD signals are conveying information directly from the incoming bit stream as required by the Huffman Decoder and Video Demultiplex circuit. Nevertheless, while the functioning of the Huffman Decoder and Video Demultiplex circuit is under the operation of the standard independent sequence of signals.

This mode of operation has been selected because it is the most efficient and could have been designed wherein special control tokens are employed for conveying the standard-dependent input to the Huffman Decoder and Video Demultiplexer instead of conveying the actual signals themselves.

20 23. INVERSE MODELLER

Inverse modeling is a feature of all three standards, and is the same for all three standards. In general, DATA tokens in the token buffer contain information about the values of the quantized coefficients, and about the number of zeros between the coefficients that are represented (a form of run length coding). The Inverse Modeller of the present invention has been adapted for use with tokens and simply expands the information about runs of zeros so that each DATA Token contains the requisite 64 values.

Thereafter, the values in the DATA Tokens are quantized coefficients which can be used by the Inverse Quantizer.

24. INVERSE QUANTIZER

15

20

25

The Inverse Quantizer of the present invention is a required element in the decoding sequence, but has been implemented in such away to allow the entire IC set to handle multi-standard data. In addition, the Inverse Quantizer has been adapted for use with tokens. The Inverse Quantizer lies between the Inverse modeller and inverse DCT (IDCT).

For example, in the present invention, an adder in the Inverse Quantizer is used to add a constant to the pel decode number before the data moves on to the IDCT.

The IDCT uses the pel decode number, which will vary according to each standard used to encode the information. In order for the information to be properly decoded, a value of 1024 is added to the decode number by the Inverse Quantizer before the data continues on to the IDCT.

Using adders, already present in the Inverse Quantizer, to standardize the data prior to it reaching the IDCT, eliminates the need for additional circuitry or software in the IC, for handling data compressed by the various standards. Other operations allowing for multistandard operation are performed during a "post quantization function" and are discussed below.

The control tokens accompanying the data are decoded and the various standardization routines that need to be performed by the Inverse Quantizer are identified in detail below. These "post quantization" functions are all implemented to avoid duplicate circuitry and to allow the IC to handle multi-standard encoded data.

25. HUPFMAN DECODER AND PARSER

Referring again to Figures 11 and 27, the Spatial Decoder includes a Huffman Decoder for decoding the data that the various compression standards have Huffman-encoded. While each of the standards, JPEG, MPEG and

15

20

25

30

H.261, require certain data to be Huffman encoded, the Huffman decoding required by each standard differs in some significant ways. In the Spatial Decoder of the present invention, rather than design and fabricate three separate Huffman decoders, one for each standard, the present invention saves valuable die space by identifying common aspects of each Huffman Decoder, and fabricating these common aspects only once. Moreover, a clever multi-part algorithm is used that makes common more aspects of each Huffman Decoder common to the other standards as well than would otherwise be the case.

In brief, the Huffman Decoder 321 works in conjunction with the other units shown in Figure 27. These other units are the Parser State Machine 322, the inshifter 323, the Index to Data unit 324, the ALU 325, and the Token Formatter 326. As described previously, connection between these blocks is governed by a two wire interface. A more detailed description of how these units function is subsequently described herein in greater detail, the focus here is on particular aspects of the Huffman Decoder, in accordance with the present invention, that support multi-standard operation.

The Parser State Machine of the present invention, is a programmable state machine that acts to coordinate the operation of the other blocks of the Video Parser. In response to data, the Parser State Machine controls the other system blocks by generating a control word which is passed to the other blocks, side by side with the data, upon which this control word acts. Passing the control word alongside the associated data is not only useful, it is essential, since these blocks are connected via a two-wire interface. In this way, both data and control arrive at the same time. The passing of the control word is indicated in Figure 27 by a control line 327 that runs

10

15

20

25

30

beneath the data line 328 that connects the blocks. Among other things, this code word identifies the particular standard that is being decoded.

The Huffman decoder 321 also performs certain control functions. In particular, the Huffman Decoder 321 contains a state machine that can control certain functions of the Index to Data 324 and ALU 325. Control of these units by the Huffman Decoder is necessary for proper decoding of block-level information. Having the Parser State Machine 322 make these decisions would take too much time.

An important aspect of the Huffman Decoder of the present invention, is the ability to invert the coded data bits as they are read into the Huffman Decoder. This is needed to decode H.261 style Huffman codes, since the particular type of Huffman code used by H.261 (and substantially by MPEG) has the opposite polarity then the codes used by JPEG. The use of an inverter, thereby, allows substantially the same table to be used by the Huffman Decoder for all three standards. Other aspects of how the Huffman Decoder implements all three standards are discussed in further detail in the "More Detailed Description of the Invention" section.

The Index to Data unit 324 performs the second part of the multi-part algorithm. This unit contains a look up table that provides the actual Huffman decoded data. Entries in the table are organized based on the index numbers generated by the Huffman Decoder.

The ALU 325 implements the remaining parts of the multi-part algorithm. In particular, the ALU handles sign-extension. The ALU also includes a register file which holds vector predictions and DC predictions, the use of which is described in the sections related to prediction filters. The ALU, further, includes counters that count through the structure of the picture being decoded by the

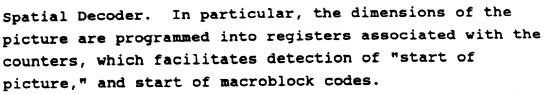
10

15

20

25

30



In accordance with the present invention, the Token Formatter 326 (TF) assembles decoded data into DATA tokens that are then passed onto the remaining stages or blocks in the Spatial Decoder.

In the present invention, the in shifter 323 receives data from a FIFO that buffers the data passing through the Start Code Detector. The data received by the inshifter is generally of two types: DATA tokens, and start codes which the Start Code Detector has replaced with their respective tokens, as discussed further in the token section. Note that most of the data will be DATA tokens that require decoding.

The ln shifter 323 serially passes data to the Huffman Decoder 321. On the other hand, it passes control tokens in parallel. In the Huffman decoder, the Huffman encoded data is decoded in accordance with the first part of the multi-part algorithm. In particular, the particular Huffman code is identified, and then replaced with an index number.

The Huffman Decoder 321 also identifies certain data that requires special handling by the other blocks shown in Figure 27. This data includes end of block and escape. In the present invention, time is saved by detecting these in the Huffman Decoder 321, rather than in the Index to Data unit 324.

This index number is then passed to the Index to Data unit 324. In essence, the Index to Data unit is a look-up table. In accordance with one aspect of the algorithm, the look-up table is little more than the Huffman code table specified by JPEG. Generally, it is in the condensed data

15

20

25

30

format that JPEG specifies for transferring an alternate JPEG table.

From the Index to Data unit 324, the decoded index number or other data is passed, together with the accompanying control word, to the ALU 325, which performs the operations previously described.

From the ALU 325, the data and control word is passed to the Token Formatter 326 (TF). In the Token Formatter, the data is combined as needed with the control word to form tokens. The tokens are then conveyed to the next stages of the Spatial Decoder. Note that at this point, there are as many tokens as will be used by the system.

26. INVERSE DISCRETE COSINE TRANSFORM

The Inverse Discrete Cosine Transform (IDCT), in accordance with the present invention, decompresses data related to the frequency of the DC component of the picture. When a particular picture is being compressed, the frequency of the light in the picture is quantized, reducing the overall amount of information needed to be stored. The IDCT takes this quantized data and decompresses it back into frequency information.

The IDCT operates on a portion of the picture which is 8x8 pixels in size. The math which performed on this data is largely governed by the particular standard used to encode the data. However, in the present invention, significant use is made of common mathematical functions between the standards to avoid unnecessary duplication of circuitry.

Using a particular scaling order, the symmetry between the upper and lower portions of the algorithms is increased, thus common mathematical functions can be reused which eliminates the need for additional circuitry. The IDCT responds to a number of multi-standard tokens. The first portion of the IDCT checks the entering data to ensure that the DATA tokens are of the correct size for processing. In fact, the token stream can be corrected in some situations if the error is not too large.

27. BUFFER MANAGER

5

10

15

20

25

The Buffer Manager of the present invention, receives incoming video information and supplies the address generators with information on the timing of the datas arrival, display and frame rate. Multiple buffers are used to allow changes in both the presentation and display rates. Presentation and display rates will typically vary in accordance with the data that was encoded and the monitor on which the information is being displayed. arrival rates will generally vary according to errors in encoding, decoding or the source material used to create the data. When information arrives at the Buffer Manager, it is decompressed. However, the data is in an order that is useful for the decompression circuits, but not for the particular display unit being used. When a block of data enters the Buffer Manager, the Buffer Manager supplies information to the address generator so that the block of data can be placed in the order that the display device can In doing this, the Buffer Manager takes into account the frame rate conversion necessary to adjust the incoming data blocks so they are presentable on the particular display device being used.

In the present invention, the Buffer Mnager primarily supplies information to the address generators.

Nevertheless, it is also required to interface with other elements of the system. For example, there is an interface with an input FIFO which transfers tokens to the Buffer Manager which, in turn, passes these tokens on to the write

address generators.

The Buffer Manager also interfaces with the display address generators, receiving information on whether the display device is ready to display new data. The Buffer Manager also confirms that the display address generators have cleared information from a buffer for display.

The Buffer Manager of the present invention keeps track of whether a particular buffer is empty, full, ready for use or in use. It also keeps track of the presentation number associated with the particular data in each buffer. In this way, the Buffer Manager determines the states of the buffers, in part, by making only one buffer at a time ready for display. Once a buffer is displayed, the buffer is in a "vacant" state. When the Buffer Manager receives a PICTURE_START, FLUSH, valid or access token, it determines the status of each buffer and its readiness to accept new data. For example, the PICTURE_START token causes the Buffer Manager to cycle through each buffer to find one which is capable of accepting the new data.

The Buffer Manager can also be configured to handle the multi-standard requirements dictated by the tokens it receives. For example, in the H.261 standard, data maybe skipped during display. If such a token arrives at the Buffer Mnager, the data to be skipped will be flushed from the buffer in which it is stored.

Thus, by managing the buffers, data can be effectively displayed according to the compression standard used to encode the data, the rate at which the data is decoded and the particular type of display device being used.

25

20

10

15

The foregoing description is believed to adequately describe the overall concepts, system implementation and operation of the various aspects of the invention in sufficient detail to enable one of ordinary skill in the art to make and practice the invention with all of its attendant features, objects and advantages. However, in order to facilitate a further, more detailed in depth understanding of the invention, and additional details in connection with even more specific, commercial implementation of various embodiments of the invention, the following further description and explanation is proferred.

This is a more detailed description for a multi-standard video decoder chip-set. It is divided into three main sections: A, B and C.

Again, for purposes of organization, clarity and convenience of explanation, this additional disclosure is set forth in the following sections.

Description of features common to chips in the chip-set:

- · Tokens
- 10 · Two wire interfaces
 - · DRAM interface
 - · Microprocessor interface
 - ·Clocks
 - · Description of the Spatial Decoder chip
 - · Description of the Temporal Decoder chip

SECTION A.1

The first description section covers the majority of the electrical design issues associated with using the chip-set.

20 A.1.1 Typographic conventions

A small set of typographic conventions is used to emphasize some classes of information:

NAMES OF TOKENS

wire name active high signal

25 wire_name active low signal
 register name

SECTION A.2 Video Decoder Family

- · 30 MHz operation
- Decodes MPEG, JPEG & H.261
- · Coded data rates to 25 Mb/s
- 5 Video data rates to 21 MB/s
 - MPEG resolutions up to 704 \times 480, 30 Hz, 4:2:0
 - · Flexible chroma sampling formats
 - Full JPEG baseline decoding
 - · Glue-less page mode DRAM interface
- 10 · 208 pin PQFP package
 - · Independent coded data and decoder clocks
 - Re-orders MPEG picture sequence

The Video decoder family provides a low chip count solution for implementing high resolution digital video decoders. The chip-set is currently configurable to support three different video and picture coding systems: JPEG, MPEG and H.261.

Full JPEG baseline picture decoding is supported. 720 \times 480, 30 Hz, 4:2:2 JPEG encoded video can be decoded in real-time.

CIF (Common Interchange Format) and QCIF H.261 video can be decoded. Full feature MPEG video with formats up to 740×480 , 30 Hz, 4:2:0 can be decoded.

Note: The above values are merely illustrative, by way of example and not necessarily by way of limitation, of one embodiment of the present invention. Accordingly, it will be appreciated that other values and/or ranges may be used.

A.2.1 System configurations

A.2.1.1 Output formatting

In each of the examples given below, some form of output formatter will be required to take the data presented at the output of the Spatial Decoder or Temporal Decoder and

15

20

25

30

re-format it for a computer or display system. The details of this formatting will vary between applications. In a simple case, all that is required is an address generator to take the block formatted data output by the decoder chip and write it into memory in a raster order.

The Image Formatter is a single chip VLSI device providing a wide range of output formatting functions.

A.2.1.2 JPEG still picture decoding

A single Spatial Decoder, with no-off-chip DRAM, can rapidly decode baseline JPEG images. The Spatial Decoder will support all features of baseline JPEG. However, the image size that can be decoded may be limited by the size of the output buffer provided by the user. The characteristics of the output formatter may limit the chroma sampling formats and color spaces that can be supported.

A.2.1.3 JPEG video decoding

Adding off-chip DRAMs to the Spatial Decoder allows it to decode JPEG encoded video pictures in real-time. The size and speed of the required buffers will depend on the video and coded data rates. The Temporal Decoder is not required to decode JPEG encoded video. However, if a Temporal Decoder is present in a multi-standard decoder chip-set, it will merely pass the data through the Temporal Decoder without alteration or modification when the system is configured for JPEG operation.

A.2.1.4 H.261 decoding

The Spatial Decoder and the Temporal Decoder are both required to implement an H.261 video decoder. The DRAM interfaces on both devices are configurable to allow the quantity of DRAM required for proper operation to be reduced when working with small picture formats and at low coded data rates. Typically, a single 4Mb (e.g. 512k x 3) DRAM will be required by each of the Spatial Decoder and

the Temperal Decoder.

A.2.1.5 MPEG decoding

The configuration required for MPEG operation is the same as for H.261. However, as will be appreciated by one of ordinary skill in the art, larger DRAM buffers may be required to support the larger picture formats possible with MPEG.

SECTION A.3 Tokens

A.3.1 Token format

5

10

15

20

25

In accordance with the present invention, tokens provide an extensible format for communicating information through the decoder chip-set. While in the present invention, each word of a Token is a minimum of 8 bits wide, one of ordinary skill in the art will appreciate that tokens can be of any width. Furthermore, a single Token can be spread over one or more words; this is accomplished using an extension bit in each word. The formats for the tokens are summarized in Table A.3.1.

The extension bit indicates whether a Token continues into another word. It is set to 1 in all words of a Token except the last one. If the first word of a Token has an extension bit of 0, this indicates that the Token is only one word long.

Each Token is identified by an Address Field that starts in bit 7 of the first word of the Token. The Address Field is of variable length and can potentially extend over multiple words (in the current chips no address is more than 8 bits long, however, one of ordinary skill in the art will again appreciate that addresses can be of any length).

Some interfaces transfer more than 8 bits of data. For example, the output of the Spatial Decoder is 9 bits wide (10 bits including the extension bit). The only Token that takes advantage of these extra bits is the DATA Token. The DATA Token can have as many bits as are necessary for carrying out processing at a particular place in the system. All other Tokens ignore the extra bits.

15

20

25

30



A.3.2 The DATA Token

The DATA Token carries data from one processing stage to the next. Consequently, the characteristics of this Token change as it passes through the decoder. Furthermore, the meaning of the data carried by the DATA Token varies depending on where the DATA Token is within the system, i.e., the data is position dependent. In this regard, the data may be either frequency domain or Pel domain data depending on where the DATA Token is within the Spatial Decoder. For example, at the input of the Spatial Decoder, DATA Tokens carry bit serial coded video data packed into 3 bit words. At this point, there is no limit to the length of each Token. In contrast, however, at the output of the Spatial Decoder each DATA Token carries exactly 64 words and each word is 9 bits wide.

A.3.3 Using Token formatted data

In some applications, it may be necessary for the circuitry that connect directly to the input or output of the Decoder or chip set. In most cases it will be sufficient to collect DATA Tokens and to detect a few Tokens that provide synchronization information (such as PICTURE_START). In this regard, see subsequent sections A.16, "Connecting to the output of Spatial Decoder", and A.19, "Connecting to the output of the Temporal Decoder".

As discussed above, it is sufficient to observe activity on the extension bit to identify when each new Token starts. Again, the extension bit signals the last word of the current token. In addition, the Address field can be tested to identify the Token. Unwanted or unrecognized Tokens can be consumed (and discarded) without knowledge of their content. However, a recognized token causes an appropriate action to occur.

Furthermore, the data input to the Spatial Decoder can either be supplied as bytes of coded data, or in DATA Tokens (see Section A.10, "Coded data input"). Supplying Tokens via the coded data port or via the microprocessor interface allows many of the features of the decoder chip set to be configured from the data stream. This provides an alternative to doing the configuration via the micro processor interface.



2 6 5	1 4	1 3	3 :	2	1 (Token Namè	So/our
2:0:1		<u></u> -	$\frac{1}{1}$	+			Reference
0, 1 0		\dotplus	 	+	1	QUANT_SCALE	
C: 1 1	<u> </u>		1		<u> </u>	PREDICTION_MODE	
	-	<u> </u>	 	+	$\frac{\perp}{}$	(reserved)	
1:00	'	<u> </u>	1			MVD_FORWARDS	
1 0 1	<u>!</u>		1	1	<u> </u>	MVD_BACKWARDS	
0:00	<u> </u>	1		<u> </u>		QUANT_TABLE	
0:00	0	0	1			DATA	
1 1 0	0	0	0			COMPONENT_NAME	
1:10	0	0	1			DEFINE_SAMPLING	
1 1 0	0	1	0			JPEG_TABLE_SELECT	
1 1 0	0	1	1		I	MPEG_TABLE_SELECT	
1 1 0	1	0	0			TEMPORAL_REFERENCE	
1 1 0	1	0	1			MPEG_DCH_TABLE	
1, 1 0	1	1	0	Ť	T	(reserved)	
1 1 0	1	1	1		Ť	(reserved)	
1 1 1 1	0	0	0	0	!	(reserved) SAVE_STATE	
1 1 1	0	0	0	1	†	(reserved) RESTORE_STATE	
1:11	0	0	1	0	i	TIME_CODE	
1 1 1 1	0	0	1	1		(reserved)	
0,00	0	0	0	0	0	NULL	
3.0 0	0	0	0	0	1	(reserved)	
0:00	0	0	0		0	(reserved)	
0,010	0	0	0	1	1	(reserved)	
0:00	1	0	0			SEQUENCE_START	
	1	!	0	0		GROUP_START	
0:0 0	1	0	0	1		PICTURE_START	
		-	!				
0:000	1	0	0	1	1	SECUENCE END	
	÷		1	0		SEQUENCE_END	
	1	0	1	0		CODING_STANDARD	
0:0		0	1	1		PICTURE_END	
0.000			1	1	1	FLUSH	
c; oj oj	1	1	0	0		FIELD_INFO	

Table A.3.1 Summary of Tokens



7 6 5 4 3 2 1 0	Token Name	Reference
0 0 0 1 1 0 0 1	MAX_COMP_ID	
0 0 0 0 1 1 0 1 0	EXTENSION_DATA	
0 0 0 1 1 0 1 1	USER_DATA	
0 0 0 0 1 1 1 0 0	DHT_MARKER	
0 0 0 1 1 1 0 1	DQT_MARKER	
0 0 0 1 1 1 1 0	(reserved) DNL_MARKER	
0 0 0 0 1 1 1 1 1 1	(reserved) DRI_MARKER	
1 1 1 0 1 0 0 0	(raserved)	
1 1 1 0 1 0 0 1	(reserved)	
1; 1 1 0 1 0 1 0	(reserved)	
1 1 1 0 1 0 1 1	(reserved)	
1 1 1 0 1 1 0 0	BIT_RATE	
1 1 1 0 1 1 0 1	VBV_BUFFER_SIZE	
1 1 1 0 1 1 1 0	VBV_DELAY	
1 1 1 0 1 1 1 1	PICTURE_TYPE	
1	PICTURE_RATE	
1:1:1 1 0 0 0 1	PEL_ASPECT	
1; 1 1 1 0 0 1 0	HORIZONTAL_SIZE	
1; 1 1 1 0 0 1 1	VERTICAL_SIZE	
1 1 1 1 0 1 0 0	BROKEN_CLOSED	
1 1 1 1 0 1 0 1	CONSTRAINED	
1 1 1 1 0 1 1 0	(reserved) SPECTRAL_LIMIT	
1 1 1 1 0 1 1 1	DEFINE_MAX_SAMPLING	
1 1 1 1 1 0 0 0	(reserved)	
1 1 1 1 1 0 0 1	(reserved)	
1 1 1 1 1 0 1 0	(reserved)	
1 1 1 1 1 0 1 1	(reserved)	
1 1 1 1 1 1 0 0	HORIZONTAL_MBS	
1:1 1 1 1 1 0 1	VERTICAL_MBS	
1 1 1 1 1 1 0	(reserved)	
1 1 1 1 1 1 1 1 1 1	(reserved)	

Table A.3.1 Summary of Tokens (contd)

A.3.4 Description of Tokens

This section documents the Tokens which are implemented in the Spatial Decoder and the Temporal Decoder chips in accordance with the present invention; see Table A.3.2.

5 Note:

- ."r" signifies bits that are currently reserved and carry the value $\ensuremath{\text{0}}$
- .unless indicated all integers are unsigned

-	
E 7 6 5 4 3 2 1	1 Castiption
1 1 1 1 1 0 1 1 1 0 1	BIT_RATE test into only
1 0 0 0 0 0 0 0	Carries the MPEG bit rate parameter P. Generated by the Huttman
0 0 0 0 0 0 0 0	decoder when decoding an MPEG bitstream.
	b - an 18 bit integer as defined by MPEG
1 1 1 1 1 1 0 1 0 0	
	╡
0	Carries two MPEG flags bits:
	c · closed_gop
1 0 0 0 1 1 0 1 1 0 1	b · broken_link
11	1
0 5 5 5 5 5 5 5	s - an 8 bit integer indicating the current coding standard. The
	values currently assigned are:
	0 - H.251
	V-n251
	1 - JPEG
	2 · MPEG
1 1 1 0 0 0 0 0 0 0 0	COMPONENT_NAME
	C
	Communicates the relationship between a component ID and the
	component name. See also
	c - 2 bit component ID
1 1 1 1 1 0 1 0 1	n - 8 bit component 'name'
	CONSTRAINED
0 , , , , , , , , ,	c - carnes the constrained_parameters_flag decoded from an
	MPEG bitstream.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 1 of 9)

Ξ

-

-

-



E 7 6 5 4 3 2 1 0 Description					
	DATA				
Carries data through the decoder chip-set.	Carries data through the decoder chip-set.				
0 d d d d d d d d c - a 2 bit integer component ID (see A.3.5.1	c - a 2 bit integer component ID (see A.3.5.1) This field				
is not defined for Tokens that carry doded data (rather that	n pixel				
information).	information).				
1 1 1 1 1 0 1 1 1 DEFINE_MAX_SAMPLING					
1 r r r r h h Max. Honzontal and Vertical sampling numbers. These de	cccha				
the maximum number of blocks horizontally/vertically in ar	ry				
component of a macroblock. See A.3.5.2					
h - 2 bit horizontal sampling number.					
v - 2 bit vertical sampling number.					
1 1 1 0 0 0 1 c c DEFINE_SAMPLING	DEFINE_SAMPLING				
1 r r r r r h h Horizontal and Vertical sampling numbers for a particular of	siour				
0 r r r r r v v component. See A.3.5.2	! !				
c - 2 bit component ID.					
h - 2 bit horizontal sampling number.					
v - 2 bit vertical sampling number.					
0 0 0 0 1 1 1 0 0 DHT_MARKER					
This Token informs the Video Demux that the DATA Toke	n stat				
follows contains the specification of a Huffman table descr	ibed				
using the JPEG "define Huffman table segment" syntax. Th	is Token				
is only valid when the coding standard is configured as JP	EG.				
This Token is generated by the start code detector during.	JPEG :				
decoding when a DHT marker has been encountered in th	e dala				

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 2 of 9)

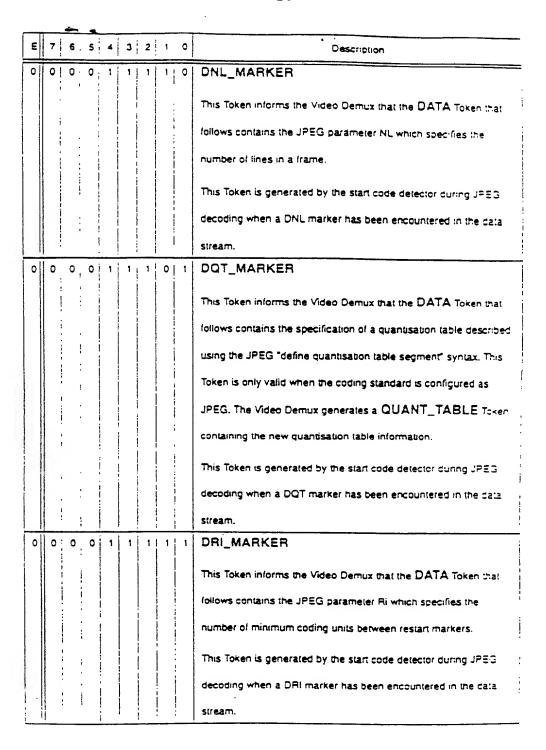


Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 3 of 9)



1 .	8 3	4	3	2	1	0	Description
1 0	0 0	<u> </u> 1	_	0	_		EXTENSION_DATA JPEG
	v v	·	٠	V	v	V	
					!		This Token informs the Video Demux that the DATA Token that
					į		follows contains extension data. See A.11 3, "Conversion of start
							codes to Tokens*, and A.14.6, "Receiving User and
							Extension data",
			Ţ,				Ouring JPEG operation the 8 bit field ❤ carries the JPEG marker
			18				value. This allows the class of extension data to be identified.
0 0	0;0	1	1	0	1	0	EXTENSION_DATA MPEG
			 				This Token informs the Video Demux that the DATA Token that
							follows contains extension data. See A.11.3, *Conversion of start
						113	codes to Tokens*, and A.14.6, "Receiving User and
							Extension data*,
1 0	0 0	1	1	0	0	0	FIELD_INFO
0 (1 1	1	p	1	1	1	Carries information about the picture following to aid its display.
							This function is not signalled by any existing coding standard.
							t - if the picture is an interlaced frame this bit indicates if the upper
							field is first (t=0) or second.
							p - if pictures are fields this indicates if the next picture is upper
							(p≖0) or lower in the frame.
							f - a 3 bit number indicating position of the field in the 8 field PAL
						; 	sequence.
0 0	0 0	1	0	j 1	1	1	FLUSH
				1			Used to indicate the end of the current coded data and to push the
·				1		 	end of the data stream through the decoder.
0 0	0 0	1	0	٥	0	1	GROUP_START
			1		:	!	Generated when the group of pictures start code is found when
				:		1	decoding MPEG or the frame marker is found when decoding
i				:		İ	JPEG.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 4 of

П	-			- 1				ī					
E	7	5	5				1		Description				
1		1	<u> </u>						HORIZONTAL_MBS				
1							n¦		h - a 13 bit number integer indicating the horizontal width of the				
٥	h	h	p	h	μ	ħ	ni	n	picture in macroblocks.				
1	1	1	1	1	0	0	1	0	HORIZONTAL_SIZE				
1	h	h	h	h	n	h j	ħ	7	h - 16 bit number integer indicating the horizontal width of the				
0	h	hi	h	h	h	n	h	'n	11 - 19 Dit Hattibet titleden wordening ale Housening Main of the				
				_	- }		j		picture in pixels. This can be any integer value.				
1	1 ;	1 }	٥j	0 ;	1	0	c¦	С	JPEG_TABLE_SELECT				
0	•	(,	1	1	1	t	1	Informs the inverse quantiser which quantisation table to use on				
									the specified colour component				
									c - 2 bit component ID (see A.3.5.1				
					1				t - 2 bit integer table number.				
1	0	0	0	1	1	0	0	1	MAX_COMP_ID				
0	1	١	•	ſ	′	,	т	m	m - 2 bit integer indicating the maximum value of component ID				
									(see A.3.5.1) that will be used in the next picture.				
0	1	1	0	; 1	0	1	c	С	MPEG_DCH_TABLE				
0	'	,	'	1	1	1	ι	t	Configures which DC coefficient Huffman table should be used for				
									colour companent cc.				
									c - 2 bit component ID (see A.3.5.1				
									t - 2 bit integer table number.				
0	1	1	0	0	1	1	٥	n	MPEG_TABLE_SELECT				
									Informs the inverse quantiser whether to use the default or user				
						1			defined quantisation table for intra or non-intra information.				
			1						n - 0 indicates intra information, 1 non-intra.				
		;	;			-	{		d - 0 indicates default table, 1 user defined.				

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 5 of 9)



								170							
Ε	7	6	5	4	3	2	1	0	Description						
1	ī	Q T	1	d	٧	٧	٧	٧	MVD_BACKWARDS						
0	~	٧	٧	٧	٧	*	*	~	Carnes one component (either vertical or horizontal) of the backwards motion vector.						
									d - 0 indicates x component, 1 the y component v - 12 bit two's complement number. The LSB provides half pixel						
									v - 12 bit two's complement number. The LSB provides half pixel resolution.						
1	1	0	0	d	٧	٧	٧	٧	MVD_FORWARDS						
o	\ •	٧	٧	٧	٧	٧	٧	٧	Carries one component (either vertical or horizontal) of the lorwards motion vector.						
									d - 0 indicates x component, 1 the y component v - 12 bit two's complement number. The LSB provides half pixel						
0	0	0	0	0	0	0	0	0	resolution. NULL						
									Does nothing.						
<u> </u>	11	1	1	1	0	0	0	1	PEL_ASPECT						
0	!	,	 ;	r	p	p	р	p	p - a 4 bit integer as defined by MPEG.						
٥١	0	c	0	1	0	1	-,	٥	PICTURE_END						
									inserted by the start code detector to indicate the end of the current picture.						
1	1	1	1	1	0	0	0	0	PICTURE_RATE						
0			1	•	p	٥	p	p	p - a 4 bit integer as defined by MPEG.						
1	o	Q	0	1	0	0	1	0	PICTURE_START						
0	,	•	٢	î	п	រា	n	n	Indicates the start of a new picture.						
-									n - a 4 bit picture index allocated to the picture by the start code detector.						

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 6 of 9)



Ε	7	6	-	5	٦	3	:	2	1	0	Description
1	1	1	i	1	0	1	i	1	1	1	PICTURE_TYPE MPEG
0	,	ſ	-	rj	٢	r	İ	•	p	D	p - a 2 bit integer indicating the picture coding type of the picture
		1	1	İ	1				i		that follows:
) 						Ì			0 - Intra
											1 - Predicted
											2 - Bidirectionally Predicted
				ì							3 - DC Intra
1	1	; .	ij	1	0		1	1	1	1	PICTURE_TYPE H.261
1	٢	1 1	·	r	t		۱	1	0	1	Indicates various H.261 options are on (1) or off (0). These options
0	1		.	s	đ		1	q	1	1	are always off for MPEG and JPEG:
											s - Split Screen Indicator
					<u> </u>						d - Document Camera
											I - Freeze Picture Release
							1				Source picture format:
		1				İ					q = 0 - QCIF
										!	q = 1 · CIF
0	ii	0	1	0	T	1	у	x	b	!	PREDICTION_MODE
		!									A set of flag bits that indicate the prediction mode for the
											macrobiocks that follow:
		1									f - forward prediction
											b - backward prediction
					-						x - reset forward vector predictor
										į	y - reset backward vector predictor
											h - enable H.261 loop filter
	0	0		 	1	s	5	5		:	QUANT_SCALE
			; ;	1	1						Informs the inverse quantiser of a new scale factor
			1		İ						s - 5 bit integer in range 1 31. The value 0 is reserved.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 7 of

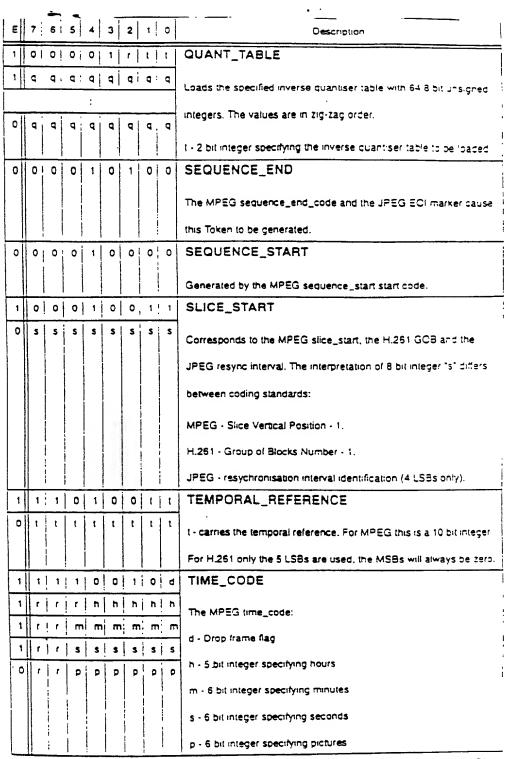


Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 8 of

-	•	•						
E! 7	5 .	5	4	3	2	;	0	Description
1 0	٥·	٥.	1	1	0.	1	1	USER_DATA JPEG
3	v	٧	٧	٧	٧	٧	٧	This Token informs the Video Demux that the DATA Token mat
	1							follows contains user data. See A.11.3, "Conversion of start codes
								to Tokens", and A.14.6, "Receiving User and
The state of the s								Extension data*,
								During JPEG operation the 8 bit field 1/ pairnes the UPEG marker
<u> </u>	ļ		!					value. This allows the class of user data to be identified
0 0	ο.	J	1	1 1	0	, 1.	1	USER_DATA MPEG
	,			!				This Token informs the Video Demux that the DATA Token that
	: !		!	:	!			follows contains user data. See A.11.3. "Conversion of start codes
i	:			•	i			to Tokens*, and A.14 5, "Receiving User and
		1		:	!	<u>:</u>	;	Extension data*,
1 1	1	1	10	1	1	a	! 1	VBV_BUFFER_SIZE
1111	ť	; ,	1	1,	1	s	s	s - a 10 bit integer as defined by MPEG.
ol s	5	s	s	s	5	5	s	
11 1	1	1	0	1	1	1	. 0	VBV_DELAY
1 6	٥	D	0	b	٥	b	þ	b - a 16 bit integer as defined by MPEG.
0 6.	b	b	6	6	b	Ь	ا ا	
1 1 1	1	! 1	1	1	j 1	; 0	1 1	VERTICAL_MBS
1 1	1	1	i v	iv	\ v	Īv	' v	v - a 13 bit integer indicating the vertical size of the picture in
oj v	٧	į v	į	~	*	į	! V	
1 1 1	1	. 1	. 1	0) 0	; 1	1 1	VERTICAL_SIZE
11				iv		; ,		
0 0	·			· ; \	,	· · v	, ,	v - a 16 bit integer indicating the vertical size of the picture in pixels
				:	<u>:</u>	<u>;</u>	:	This can be any integer value.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 9 of 9)

A.3.5 Numbers signalled in Tokens

A.3.5.1 Component Identification number

In accordance with the present invention, the Component ID number is a 2 bit integer specifying a color component. This 2 bit field is typically located as part of the Header in the DATA Token. With MPEG and H.261 the relationship is set forth in Table A.3.3.

Component ID	MPEG or H.261 colour companent
0	Luminance (Y)
1	Blue difference signal (Cb / U)
2	Red difference signal (Cr / V)
3	Never used

Table A.3.3 Component ID for MPEG and H.261

With JPEG the situation is more complex as JPEG does not limit the color components that can be used. The decoder chips permit up to 4 different color components in each scan. The IDs are allocated sequentially as the specification of color components arrive at the decoder.

A.3.5.2 Horizontal and Vertical sampling numbers

For each of the 4 color components, there is a specification for the number of blocks arranged horizontally and vertically in a macroblock. This specification comprises a two bit integer which is one less

For example, in MPEG (or H.261) with 4:2:0 chroma sampling (Figure 36) and component IDs allocated as per Table A.3.4.

than the number of blocks.

	Honzontal		Vertical	
Component ID	sampling	Width in blocks	sampling	Height in blocks
	number		number	
0	1	2	1	2
1	0	1	0	1
2	0	1	0	t
3	Not used	Not used	Not used	Not used

Table A.3.4 Sampling numbers for 4:2:0/MPEG

15

With JPEG and 4:2:2 chroma sampling (allocation of component to component ID will vary between applications. See A.3.5.1. Note: JPEG requires a 2:1:1 structure for its macroblocks when processing 4:2:2 data. See Table A.3.5.

Component ID	Honzontal sampling number	Width in blocks	Vertical sampling number	Height in blocks		
Y	1	2	0	1		
υ	0	1	0	1		
V	0	1	0	1		

Table A.3.5 Sampling numbers for 4:2:2 JPEG

A.3.6 Special Token formats

In accordance with the present invention, tokens such as the DATA Token and the QUANT_TABLE Token are used in their "extended form" within the decoder chip-set. In the extended form the Token includes some data. In the case of DATA Tokens, they can contain coded data or pixel data. In the case of QUANT_TABLE tokens, they contain quantizer table information.

Furthermore, "non-extended form" of these Tokens is defined in the present invention as "empty". This Token format provides a place in the Token stream that can be subsequently filled by an extended version of the same Token. This format is mainly applicable to encoders and, therefore, it is not documented further here.

Token Name	MPEG	JPEG	H.251
BIT_RATE	1		
BROKEN_CLOSED	1		
CODING_STANDARD	1	1	/
COMPONENT_NAME		1	
CONSTRAINED	1		
DATA	1	1	1
DEFINE_MAX_SAMPLING	1	1	1
DEFINE_SAMPLING	1	1	1
DHT_MARKER		1	
DNL_MARKER		1	
DQT_MARKER		1	
DRI_MARKER .		1	

Table A.3.6 tokens for different standards

15



Token Name	MPEG	JPEG	H.251
EXTENSION_DATA	1	/	
FIELD_INFO			
FLUSH	1	1	/
GROUP_START	1.	1	
HORIZONTAL_MBS	1	1	1
HORIZONTAL_SIZE	1	1	1
JPEG_TABLE_SELECT		/	
MAX_COMP_ID	/	1	1
MPEG_DCH_TABLE	1		
MPEG_TABLE_SELECT	1		
MVD_BACKWARDS	1		
MVD_FORWARDS	/		/
NULL	1	1	/
PEL_ASPECT	1		
PICTURE_END	1	1	1
PICTURE_RATE	1		
PICTURE_START	1	1	1
PICTURE_TYPE	1	1	1
PREDICTION_MODE	1		
QUANT_SCALE	1		/
QUANT_TABLE	1	1	
SEQUENCE_END	1	1	
SEQUENCE_START	1	/	1
SLICE_START	1	1	1
TEMPORAL_REFERENCE	1		1
TIME_CODE	1		
USER_DATA	1	1	
VBV_BUFFER_SIZE	1		
VBV_DELAY	1		
VERTICAL_MBS	/	/	1
VERTICAL_SIZE	/	/	/

Table A.3.6 Tokens for different standards (contd)

A.3.7 Use of Tokens for different standards Each standard uses a different sub-set of the defined Tokens in accordance with the present invention; ss Table A.3.6.

SECTION A.4 The two wire interface

A.4.1 Two-wire interfaces and the Token Port

A simple two-wire valid/accept protocol is used at all levels in the chip-set to control the flow of information. Data is only transferred between blocks when both the sender and receiver are observed to be ready when the clock rises.

- 1) Data transfer
- 2) Receiver not ready
- 3) Sender not ready

5

15

If the sender is not ready (as in 3 Sender not ready above) the input of the receiver must wait. If the receiver is not ready (as in 2 Receiver not ready above) the sender will continue to present the same data on its output until it is accepted by the receiver.

When Token information is transferred between blocks the two-wire interface between the blocks is referred to as a Token Port.

A.4.2 Where used

The decoder chip-set, in accordance with the present invention, uses two-wire interfaces to connect the three chips. In addition, the coded data input to the Spatial Decoder is also a two-wire interface.

A.4.3 Bus signals

The width of the data word transferred by the two-wire interface varies depending upon the needs of the interface concerned (See Figure 35, "Tokens on interfaces wider than 8 bits". For example, 12 bit coefficients are input to the Inverse Discrete Cosine Transform (IDCT), but only 9 bits are output.

Interface	Data Width (bits)
Coded data input to Spanal Decoder	8
Cutput port of Spatial Decoder	Э
Input part of Temporal Decoder	9
Output port of Temporal Decoder	8
input port of image Formatter	ā

Table A.4.1 Two wire interface data width

In addition to the data signals there are three other signals transmitted via the two-wire interface:

- .valid
- 5 .accept

15

.extension

A.4.3.1 The extension signal

The extension signal corresponds to the Token extension bit previously described.

10 A.4.4 Design considerations

The two wire interface is intended for short range, point to point communication between chips.

The decoder chips should be placed adjacent to each other, so as to minimize the length of the PCB tracks between chips. Where possible, track lengths should be kept below 25 mm. The PCB track capacitance should be kept to a minimum.

The clock distribution should be designed to minimize the clock slew between chips. If there is any clock slew, it should be arranged so that "receiving chips" see the clock before "sending chips".

All chips communicating via two wire interfaces should operate from the same digital power supply.

A.4.5 Interface timing

Nium		30	30 MHz		
Num. Chara	Characteristic	Min.	Max.	Unit	5
l	Input signal set-up time	5		ns	
2	Input signal hold time	0		ns	
3	Output signal drive time		23	ns	<u> </u>
4	Output signal hold time	. 2		ns	

Table A.4.2 Two wire interface timing

- a. Figures in Table A.4.2 may vary in accordance with design variations
 - b. Maximum signal loading is approximately 20 F

10

Note: Figure 38 shows the two-wire interface between the system de-mux chip and the coded data port of the Spatial Decoder operating from the main decoder clock. This is optional as this two wire interface can work from the coded data clock which can be asynchronous to the decoder clock. See Section A.10.5, "Coded data clock". Similarly the display interface of the Image Formatter can operate from a clock that is asynchronous to the main decoder clock.

A.4.6 - Signal levels

The two-wire interface uses CMOS inputs and output. $V_{\rm 1Hmin}$ is approx. 70% of $V_{\rm DD}$ and $V_{\rm Hmin}$ is approx. 30% of $V_{\rm DD}$. The values shown in Table A.4.3 are those for $V_{\rm 1H}$ and $V_{\rm H}$ at their respective worst case $V_{\rm DD}$. $V_{\rm 2D}$ =5.0±0.25V.

Symbol	Parameter	Min.	Max.	Units
V _{'14}	Input logic "1" voltage	3.68	V ₃₀ + 0.5	· v
٧,	Input log-c 'O' voltage	GND - 0.5	1 43	. 17
V2H	Output logic '1' voltage	V ₂₀ - 0.1	1	12.4
		V ₂₀ - 0.4	İ	, Va
You	Output topic '0' voltage		0.1	. 1=
			0.4	[y a
l _{IM}	Input leakage current		= 10	<u>ئ</u> ے :

Table A.4.3 DC electrical characteristics

- a. l_{oH}≤lmA
- b. l_{OH}≤4mA
- $c: l_{\alpha_i} \leq 1 mA$
- 10 d. $l_{O_1} \leq 4 \text{ mA}$

A.4.7 _ Control clock

In general, the clock controlling the transfers across the two wire interface is the chip's decoder_clock. The exception is the coded data port input to the Spatial Decoder. This is controlled by coded_clock. The clock signals are further described herein.

SECTION A.5 DRAM Interface

A.5.1 The DRAM interface

A single high performance, configurable, DRAM interface is used on each of the video decoder chips. In general, the DRAM interface on each chip is substantially the same; however, the interfaces differ from one another in how they handle channel priorities. The interface is designed to directly drive the DRAM used by each of the decoder chips. Typically, no external logic, buffers or components will be necessary to connect the DRAM interface to the DRAMs in most systems.

A.5.2 Interface signals

	input/	Description
Signal Name	Output	Description
DRAM_data(31:0)	1/0	The 32 bit wide DRAM data bus. Optionally this bus
		can be configured to be 16 or 8 bits wide. See
	-	section A.5.8
ORAM_addr[10:0]	0	The 22 bit wide DRAM interface address is time
		multiplexed over this 11 bit wide bus.
PAS	0	The DRAM Row Address Strade signal
CAS(3:0)	0	The DRAM Column Address Strobe signal. One
		signal is provided per byte of the interface's data
		bus. All the CAS signals are driven simultaneously
WE	0	The DRAM Write Enable signal
ŌĒ	0	The DRAM Output Enable signal
DRAM_enable	1	This input signal, when low, makes all the output
		signals on the interface go high impedance.
		Note: on-chip data processing is not stopped when
		the DRAM interface is high impedance. So, errors
		will occur if the chip attempts to access DRAM while
		DRAM_enable is low.

Table A.5.1 DRAM interface signals

20

30

In the present invention, the interface is configurable in two ways:

.The detail timing of the interface can be configured to accommodate a variety of different DRAM types

.The "width" of the DRAM interface can be configured to provide a cost/performance trade-off in different applications.

A.5.3 Configuring the DRAM interface

10 Generally, there are three groups of registers associated with the DRAM interface: interface timing configuration registers, interface bus configuration registers and refresh configuration registers. The refresh configuration registers (registers in Table A.5.4) should be configured last.

A.5.3.1 Conditions after reset

After reset, the DRAM interface, in accordance with the present invention, starts operation with a set of default timing parameters (that correspond to the slowest mode of operation). Initially, the DRAM interface will continually execute refresh cycles (excluding all other transfers). This will continue until a value is written into refresh_interval. The DRAM interface will then be able to perform other types of transfer between refresh cycles.

25 A.5.3.2 Bus configuration

Bus configuration (registers in Table A.5.3) should only be done when no data transfers are being attempted by the interface. The interface is placed in this condition immediately after reset, and before a value is written into refresh_interval. The interface can be re-configured later, if required, only when no transfers are being attempted. See the Temporal Decoder chip_access register (A.13.3.1) and the Spatial Decoder buffer_manager_access register (A.13.1.1).

25

A.5.3.3 Interface timing configuration

In accordance with the present invention, modifications to the interface timing configuration information are controlled by the interface timing access register.

Writing 1 to this register allows the interface timing registers (in Table A.5.2) to be modified. While interface timing access = 1, the DRAM interface continues operation with its previous configuration. After writing 1, the user should wait until 1 can be read back from the interface timing access before writing to any of the interface timing registers.

When configuration is compete, 0 should be written to the interface_timing_access. The new configuration will then be transferred to the DRAM interface.

15 A.5.3.4 Refresh configuration

The refresh interval of the DRAM interface of the present invention can only be configured once following reset. Until refresh_interval is configured, the interface continually executes refresh cycles. This prevents any other data transfers. Data transfers can start after a value is written to refresh interval.

As is well known in the art, DRAMs typically require a "pause" of between 100 μ s and 500 μ s after power is first applied, followed by a number of refresh cycles before normal operation is possible. Accordingly, these DRAM start-up requirements should be satisfied before writing a value to refresh interval.

A.5.3.5 Read access to configuration registers

All the DRAM interface registers of the present invention can be read at any time.

A.5.4 Interface timing (ticks)

The=DRAM interface timing is derived from a Clock which is running at four times the input Clock rate of the device (decoder_clock). This clock is generated by an on-chip PLL.

5 For brevity, periods of this high speed clock are referred to as ticks.





A.5.5 Thterface registers

Register name	Stra/Dls.	Rusul Statu	Description
interface_timing_access	1 1	0	This function enable register allows access to
	bit		the DRAM interface timing configuration
·			registers. The configuration registers should not
	rw		be modified while this register holds the value
			Writing a one to this register requests access
			to modify the configuration registers. After a D
			has been written to this register the CRAM
			interface will start to use the new values in the
			timing configuration registers.
page_start_length	5	0	Specifies the length of the access start in ticks.
	bit		The minimum value that can be used is 4
			(meaning 4 ticks). 0 selects the maximum
	~		length of 32 ticks.
transfer_cycle_length	4	0	Specifies the length of the fast page read or
	Dit		write cycle in ticks. The minimum value that can
			be used is 4 (meaning 4 ticks). 0 selects the
	~		maximum length of 16 ticks.
refresh_cycle_length	4	0	Specifies the length of the refresh cycle in ticks.
	bit		The minimum value that can be used is 4
			(meaning 4 ticks), 0 selects the maximum
	rw		length of 16 ticks.
RAS_falling	4	0	Specifies the number of ticks after the start of
	bit		the access start that RAS falls. The minimum
			value that can be used is 4 (meaning 4 ticks). 3
	-w	-	selects the maximum length of 15 toks.
CAS_falling	4	в	Specifies the number of ticks after the start of a
	bit		read cycle, write cycle or access start that CAS
			falls. The minimum value that can be used is 1
	~		(meaning 1 tick). O selects the maximum length
			of 16 ticks.

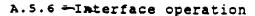
Table A.5.2 Interface timing configuration registers

Register name	Size/Dir.	Roset State	Description
DRAM_data_width	2	0	Specifies the number of bits used on the DRAM
	bit	 	interface data bus DRAM_data[31:0]. See
			A.5.8
	rw		
row_address_bits	2	0	Specifies the number of bits used for the row
	bit		address portion of the DRAM interface address
			bus. See A.5.10
	~		
DRAM_enable	1	1	Writing the value 0 in to this register forces the
	bit		DRAM interface into a high impedance state.
			0 will be read from this register if either the
	w		DRAM_enable signal is low or 0 has been
			written to the register.
CAS_strength	3	6	These three bit registers configure the output
RAS_strength	bit		drive strength of DRAM interface signals.
addr_strength	_		This allows the interface to be configured for
DRAM_data_strength	~~		vangus different loads.
OEWE_strength			ישווטעט מוויפרפתו וספסט.
			S 4.5.12
			See A.5.13

Table A.5.3 Interface bus configuration registers

...

art ri



The DRAM interface uses fast page mode. Three different types of access are supported:

- .Read
- .Write

5

.Refresh

Each read or write access transfers a burst of 1 to 64 bytes to a single DRAM page address. Read and write transfers are not mixed within a single access and each successive access is treated as a random access to a new DRAM page.

Régister name	Size/Dir.	Rosol State	Description
refresh_interval	8	0	This value specifies the interval between
	bit		refresh cycles in periods of 16 decoder_clock
			cycles. Values in the range 1255 can be
	~		configured. The value 0 is automatically loaded
			after reset and forces the DRAM interface to
			continuously execute refresh cycles until a valid
			refresh interval is configured. It is
			recommended that refresh_interval should be
			configured only once after each reset.
no_refresh	1	0	Writing the value 1 to this register prevents
	bit		execution of any refresh cycles.
	~		

Table A.5.4 Refresh configuration registers

A.5.7 -Aecess structure

Each access is composed of two parts:

- .Access start
- .Data transfer
- In the present invention, each access begins with an access start and is followed by one or more data transfer cycles. In addition, there is a read, write and refresh variant of both the access start and the data transfer cycle.
- 10 Upon completion of the last data transfer for a particular access, the interface enters its default state (see A.5.7.3) and remains in this state until a new access is ready to begin. If a new access is ready to

begin when the last access has finished, then the new access will begin immediately.

A.5.7.1 Access start

The access start provides the page address for the read or write transfers and establishes some initial signal conditions. In accordance with the present invention, there are three different access starts:

- .Start of read
- .Start of write
- .Start of refresh

Num.	Characteristic	Min.	Max.	Unit	Notes
5	RAS precharge period set by register	4	16	DCK	
	RAS_falling			Ì	
6	Access start duration set by register	4	32		
	page_start_length				
7	CAS precharge length set by register	1	16		•
	CAS_falling.	[
8	Fast page read or write cycle length set by	4	16		
	the register transfer_cycle_length.				
9	Refresh cycle length set by the register	4	16		
	refresh_cycle.				

Table A.5.5 DRAM Interface timing parameters

a. This value must be less than RAS_falling to ensure CAS before RAS refresh occurs.

In each case, the timing of RAS and the row address is controlled by the registers RAS_falling and page_start_length. The state of OE and DRAM_data[31:0] is held from the end of the previous data transfer until **RAS falls. The three different access start types only vary in how they drive OE and DRAM_data[31:0] when RAS falls. See Figure 43.

A.5.7.2 Data transfer

In the present invention, there are different types of data transfer cycles:

- .Fast page read cycle
- .Fast page late write cycle
- .Refresh cycle

A start of refresh can only be followed by a single refresh cycle. A start of read (or write) can be followed by one or more fast page read (or write) cycles. At the start of the read cycle CAS is driven high and the new column address is driven.

Furthermore, an early write cycle is used. WE is driven low at the start of the first write transfer and remains low until the end of the last write transfer. The output data is driven with the address.

As a CAS before RAS refresh cycle is initiated by the start of refresh cycle, there is no interface signal activity during the refresh cycle. The purpose of the refresh cycle is to meet the minimum RAS low period

A.5.7.3 Interface default state

The interface signals in the present invention enter a default state at the end of an access:

RAS, CAS and WE high

*data and OE remain in their previous state

.addr remains stable

A.5.8 Data bus width

required by the DRAM.

The two bit register, DRAM_data_width, allows the width of the DRAM interface's data path to be configured. This allows the DRAM cost to be minimized when working with small picture formats.

DRAM_data_width

8 bit wide data bus on DRAM_data(31:24)°.

1 16 bit wide data bus on DRAM_data(31:16)^{bi}.

2 32 bit wide data bus on DRAM_data(31:0).

Table A.5.6 Configuring DRAM_data_width

a. Default after reset.

١,

b. Unused signals are held high impedance.

A.5.9 row address width

The number of bits that are taken from the middle

section of the 24 bit internal address in order to provide

the row address is configured by the register,
row_address_bits.

row_address_bits	Width of row address
1	10 bits on DRAM_addr(9:0)
2	11 bits on DRAM_addr(10:0)

Table A.5.7 Configuring row_address_bits

A.5.10-Address bits

On-chip, a 24 bit address is generated. How this address is used to form the row and column addresses depends on the width of the data bus and the number of bits selected for the row address. Some configurations do not permit all the internal address bits to be used and, therefore, produce "hidden bits)".

Similarly, the row address is extracted from the middle portion of the address. Accordingly, this maximizes the rate at which the DRAM is naturally refreshed.

row address width	row address translation internal ⇒ external	data bus width	column address translation internal ⇔ external		
9	[14:6] ⇔ [8:0]	8	[19:15] ← [10:6]	[5:0] ⇒ [5:0]	
		16	[20:15] = [10:5]	[5:1] ⇒ [4:0]	
		32	[21:15] = [10:4]	[5:2] ⇒ [3:0]	
10	[15:6] ⇔ [9:0]	8	[19:16] - [10:6]	[5:0] ⇒ [5.0]	
		16	[20:16] 🗢 [10:5]	[5:1] ⇔ [4:0]	
		32	[21:16] ⇒ [10:4]	[5:2] \to [3-0]	
11	[16:6] => [10:0]	8	[19:17] ← [10:6]	[5:0] ⇒ [5:0]	
		16	[20:17] ⇒ [10:5]	(5:1] ⇒ [4:0]	
		32	[21:17] ⇒ [10:4]	(5.2) ⇒ (3.0)	

Table A.5.8 Mapping between internal and external addresses

A.5.14-14 Low order column address bits

The least significant 4 to 6 bits of the column address are used to provide addresses for fast page mode transfers of up to 64 bytes. The number of address bits required to control these transfers will depend on the width of the data bus (see A.5.8).

A.5.10.2 Decoding row address to access more DRAM banks

Where only a single bank of DRAM is used, the width of the row address used will depend on the type of DRAM used. Applications that require more memory than can be typically provided by a single DRAM bank, can configure a wider row address and then decode some row address bits to select a single DRAM bank.

NOTE: The row address is extracted from the middle of the internal address. If some bits of the row address are decoded to select banks of DRAM, then all possible values of these "bank select bits" must select a bank of DRAM. Otherwise, holes will be left in the address space.

A.5.11 DRAM Interface enable

In the present invention, there are two ways to make all the output signals on the DRAM interface become high impedance, i.e., by setting the DRAM_enable register and the DRAM-enable signal. Both the register and the signal must be at a logic 1 in order for the drivers on the DRAM interface to operate. If either is low then the interface is taken to high impedance.

Note: on-chip data processing is not terminated when the DRAM interface is at high impedance. Therefore, errors will occur if the chip attempts to access DRAM while the interface is at high impedance.

In accordance with the present invention, the ability to take the DRAM interface to high impedance is provided to allow other devices to test or use the DRAM controlled by the Spatial Decoder (or the Temporal Decoder) when the

30

10

Spatiant Decoder (or the Temporal Decoder) is not in use. It is not intended to allow other devices to share the memory during normal operation.

A.5.12 Refresh

5

10

15

20

Unless disabled by writing to the register, no_refresh, the DRAM interface will automatically refresh the DRAM using a CAS before RAS refresh cycle at an interval determined by the register, refresh interval.

The value in refresh_interval specifies the interval between refresh cycles in periods of 16 decoder_clock cycles. Values in the range 1.255 can be configured. The value 0 is automatically loaded after reset and forces the DRAM interface to continuously execute refresh cycles (once enabled) until a valid refresh interval is configured. It is recommended that refresh_interval should be configured only once after each reset.

While reset is asserted, the DRAM interface is unable to refresh the DRAM. However, the reset time required by the decoder chips is sufficiently short, so that it should be possible to reset them and then to re-configure the DRAM interface before the DRAM contents decay.

A.5.13 Signal strengths

The drive strength of the outputs of the DRAM interface can be configured by the user using the 3 bit registers,

CAS_strength, RAS_strength, addr_strength,

DRAM_data_strength, and OEWE_strength. The MSB of this 3 bit value selects either a fast or slow edge rate. The two less significant bits configure the output for different load capacitances.

The default strength after reset is 6 and this configures the outputs to take approximately 10ns to drive a signal between GND and V_{DD} if loaded with 24 F.

stength value	Orive characteristics
0 .	Approx. 4 ns/V into 6 pf load
1	Approx. 4 ns/V into 12 pf load
2	Approx. 4 ns/V into 24 pf load
3	Approx. 4 ns/V into 48 pf load
4	Approx. 2 ns/V into 6 pf load
5	Approx. 2 ns/V into 12 pf load
64	Approx. 2 ns/V into 24 pf load
7	Approx. 2 ns/V into 48 pf load

Table A.5.9 Output strength configurations

a. Default after reset

When an output is configured appropriately for the load it is driving, it will meet the AC electrical

characteristics specified in Tables A.5.13 to A.5.16. Wher appropriately configured, each output is approximately matched to its load and, therefore, minimal overshoot will occur after a signal transition.

A.5.14 Electrical specifications

All information provided in this section is merely illustrative of one embodiment of the present invention and is included by example and not necessarily by way of limitation.

Symbol	Parameter	Min.	Мах.	Units
∨ء٥	Supply voltage relative to GND	-0.5	6.5	V
VIN	Input voltage on any pin	GND - 0.5	Voc + 0 5	V
TA	Operating temperature	-40	+85	•0
⊤s	Storage temperature	-55	+150)*C

Table A.5.10 Maximum Ratings'

Table A.5.10 sets forth maximum ratings for the illustrative embodiment only. For this particular embodiment stresses below those listed in this table should be used to ensure reliability of operation.

Symbol	Parameter	Min.	Max.	Units
V ₀₀	Supply voltage relative to GND	4.75	5.25	١٧
GND	Ground	0	0	\ \
V _{IH}	input logic '1' voltage	2.0	V _{DO} - 0.5	V
∨ار	input logic '0' voltage	GND - 0.5	0.8	V
TA	Operating temperature	0	70	•C•

Table A.5.11 DC Operating conditions

a. With TBA linear ft/min transverse airflow

Sympol	Parameter	Min.	Max.	Units
Vol	Output logic '0' voltage		1 0.4	Y *
Voн	Output logic '1' voltage	2.8] v
lo	-Output current	± 100		د ميز
loz	Output off state leakage current	= 20		μΑ
l _{iZ}	Input leakage current	± 10		μΑ
100	RMS power supply current		500	mA .
CIN	Input capacitance		5	pF
Cout	Output / IO capacitance		5	pF

Table A.5.12 DC Electrical characteristics

- a. AC parameters are specified using $V_{\text{OLmax}} = 0.8 \text{V}$ as the measurement level.
- b. This is the steady state drive capability of the interface.

Transient currents may be much greater.

A.5.1471 AC characteristics

Num.	Parameter	Min.	Max.	Unit	Note *
10	Cycle time	-2	+2	ns	
11	Cycle time	-2	+2	ns	
12	High pulse	-5	+2	ns	
13	Low pulse	-11	+2	ns	
11	Cycle time	-8	+2	ns	

Table A.5.13 Differences from nominal values for a strobe

a. As will be appreciated by one of ordinary skill in the art, the driver strength of the signal must be configured appropriately for its load.

Num.	Parameter	Min.	Max.	Unit	Note 4
15	Strobe to strobe delay	-3	+3	ns	
16	Low hold time	-13	+3	ns	1
17	Strobe to strobe precharge e.g. (CRP,	-9	+3	ns	Ī
	IRCS, IACH, IRRH, IAPC				
	CAS precharge pulse between any two	-5	+2	ns	
	CAS signals on wide DRAMs e.g. tCP, or	ļ			
	between RAS rising and CAS falling e.g.				
	1RPC				
18	Precharge before disable	-12	+3	ns	

Table A.5.14 Differences from nominal values between two strobes

a. The driver strength of the two signals must be configured appropriately for their loads.

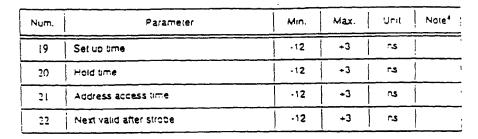


Table A.5.15 Differences from nominal between a bus and a strobe

a. The driver strength of the bus and the strobe must be configured appropriately for their loads.

Num.	Parameter	Min.	Max.	Unit	Note ,
23	Read data set-up time before CAS signal	0		.÷s	
	starts to rise	}			
24	Read data hold time after CAS signal	0		r-s	
	starts to go high				i

Table A.5.16 Differences from nominal between a bus and a strobe

When reading from DRAM, the DRAM interface samples $\overline{\text{DRAM}}$ _data[31:0] as the $\overline{\text{CAS}}$ signals rise.

parar	meter	para	parameter		parameter	
name	number	name	number	name	number	
IPC	10	IASH	16	IRHCP	18	
				ксран		
tAC	11	tCSH		IASR	19	
(AP	12	1AWL		IASC	1	
ICP		1CWL		tDS		
ICPN		tPAC		tRAH	20	
tRAS	13	tOAC/tOE		tCAH		
ICAS		1CHR		tOH		
ICAC		tCRP	17	FA		
(WP		tRCS	· ·	IAA	21	
IRASP		IRCH		tRAL		
tRASC		tRRH		IRAD	22	
tACP/ICPA	14	tRPC				
tRCD	15	tCP				
ICSA		tAPC				

Table A.5.17 Cross-reference between "standard" DRAM parameter names and timing parameter numbers

.

.

.



SECTION A.6 Microprocessor interface (MPI)

A standard byte wide microprocessor interface (MPI) is used on all chips in the video decoder chip-set. However, one of ordinary skill in the art will appreciate that microprocessor interfaces of other widths may also be used. The MPI operates synchronously to various decoder chip clocks.

A.6.1 MPI signals

Signal Name	Input/	0
Signal Name	Output	Description
enable[1:0]	Input	Two active low chip enables, Soth must be low to
		enable accesses via the MPI
~	Input	High indicates that a device wishes to read values
		from the video chip.
		This signal should be stable while the chip is
		enabled.
addr(n:0)	Input	Address specifies one of 2 ⁿ locations in the chip's
		memory map.
		This signal should be stable while the chip is
		enabled.
data(7:0]	Опфл	8 bit wide data I/O port. These pins are high
		impedance if either enable signal is high.
īrq	Output	An active low, open collector, interrupt request
		signal.

Table A.6.1 MPI interface signals

11

1.1

A.6.2 -MRI electrical specifications

Symbol	Parameter	Min.	Max.	Units
V _{D0}	Supply voltage relative to GND	-0.5	6.5	Iv
V _{IN}	Input voltage on any pin	GND - 0.5	V _{DD} + 0.5	İV
TA	Operating temperature	40	+85	•c
^T s	Storage temperature	-55	+150	•c

Table A.6.2 Absolute Maximum Ratings'

Symbol	Parameter	Min.	Max.	Units
V20	Supply voltage relative to GND	4.75	5.25	İv
GND	Ground	0	0	İv
V _{'H}	Input logic '1' voltage	2.0	V _{DO} + 0.5	V *
٧,ڔ	Input logic '0' voltage	GND - 0.5	0.8	V [8]
TA	Operating temperature	0	70	•C°

Table A.6.3 DC Operating conditions

- a. AC input parameters are measured at a 1.4V measurement level.
- b. With TBA linear ft/min transverse airflow.

Symbol	Parameter	Min.	Max.	Units
V _{OL}	Output logic 'O' voltage		0.4	V
[∨] ٥١∞	Open collector output logic '0'		0.4	V ^a
	voltage			
V _{CH}	Output logic '1' voltage	2.4		l v
lo	Output current	± 100		د میر
¹ ∞e	Open collector output current	4.0	8.0	l mA °
loz	Output off state leakage current		= 20	μΑ
l _N	Input leakage current		± 10	μΑ
loo	RMS power supply current		500	r:A
C. ₄	Input capacitance		5	⇒F
C _{CUT}	Output / IO capacitance		5	p=

Table A.6.4 DC Electrical characteristics

- a. $l_0 \le l_{0\infty \text{ min}}$.
- b. This is the steady state drive capability of the interface. Transient currents may be much greater.
- c. When asserted the open collector $\overline{\text{rq}}$ output pulls down with an impedance of 1000 or less.



A.6.2.T AC characteristics

Yum.	Characteristic	Min.	Max.	Unit	Notes
25	Enable low period	100	i	rs.	i
26	Enable high period	50	Ì	ns	İ
27	Address or rw set-up to chip enable	0	İ	ns	
28	Address or rw hold from chip disable	0		ns	1
29	Output turn-on time	20		ns	
30	Read data access time		70	ns	5
31	Read data hold time	5		ns	
32	Read data turn-off time		20		

Table A.6.5 Microprocessor interface read timing

- a. The choice, in this example, of enable[0] to start the cycle and enable[1] to end it is arbitrary. These signal are of equal status.
- b. The access time is specified for a maximum load of 50 $_{\rm p}F$ on each of the data[7.0]. Larger loads may increase the access time.

Num.	Characteristic	Min,	Max.	Unit	Notes
33	Write data set-up time	15		ns	•
3.4	Write data hold time	0		ns	

Table A.6.6 Microprocessor interface write timing

a. The choice, in this example, of enable[0] to start the cycle and enable[1] to end it is arbitrary. These signal are of equal status.

... IN NO. 1M

A.6.3 -Interrupts

5

30

In accordance with the present invention, "event" is the term used to describe an on-chip condition that a user might want to observe. An event can indicate an error or it can be informative to the user's software.

There are two single bit registers associated with each interrupt or "event". These are the condition event register and the condition mask register.

A.6.3.1 condition event register

The condition event register is a one bit read/write register whose value is set to one by a condition occurring within the circuit. The register is set to one even if the condition was merely transient and has now gone away. The register is then guaranteed to remain set to one until the user's software resets it (or the entire chip is reset).

The register is set to zero by writing the value one

Writing zero to the register leaves the register unaltered.

The register must be set to zero by user software

before another occurrence of this condition can be observed.

The register will be reset to zero on reset.

25 A.6.3.2 Condition mask register

The condition mask register is one bit read/write register which enables the generation of an interrupt request if the corresponding condition event register(s) is(are) set. If the condition event is already set when 1 is written to the condition mask register, an interrupt request will be issued immediately.

The value 1 enables interrupts.

The register clears to zero on reset.

Unless stated otherwise a block will stop operation

11 n mji 10

g 10 1 1 1 1

after seeerating an interrupt request and will re-start operation after either the condition event or the condition mask register is cleared.

A.6.3.3 Event and mask bits

- Event bits and mask bits are always grouped into corresponding bit positions in consecutive bytes in the memory map (see Table A.9.6 and Table A.17.6). This allows interrupt service software to use the value read from the mask registers as a mask for the value in the event
- registers to identify which event generated the interrupt.

A.6.3.4 The chip event and mask

Each chip has a single "global" event bit that summarizes the event activity on the chip. The chip event register presents the OR of all the on-chip events that have 1 in their mask bit.

A 1 in the chip mask bit allows the chip to generate interrupts. A 0 in the chip mask bit prevents any on-chip events from generating interrupt requests.

Writing 1 to 0 to the chip event has no effect. It will only clear when all the events (enabled by a 1 in their mask bit) have been cleared.

A.6.3.5 The irq signal

The irq signal is asserted if both the chip event bit and the chip event mask are set.

The $\overline{\text{lrq}}$ signal is an active low, "open collector" output which requires an off-chip pull-up resistor. When active the $\overline{\text{lrq}}$ output is pulled down by an impedance of 100 Ω or less.

I will be appreciated that pull-up resistor of approximately $4k\Omega$ should be suitable for most applications. A.6.4 Accessing registers

A.6.4.1 Stopping circuits to enable access

In the present invention, most registers can only

H H 1 H 1 H 1

25

30

modified if the block with which they are associated is stopped. Therefore, groups of registers will normally be associated with an access register.

The value 0 in an access register indicates that the group of registers associated with that access register should not be modified. Writing 1 to an access register requests that a block be stopped. However, the block may not stop immediately and block's access register will hold the value 0 until it is stopped.

Accordingly, user software should wait (after writing 1 to request access) until 1 is read from the access register. If the user writes a value to a configuration register while its access register is set to 0, the results are undefined.

15 A.6.4.2 Registers holding integers

The least significant bit of any byte in the memory map is that associated with the signal data[0].

Registers that hold integers values greater than 8 bits are split over either 2 or 4 consecutive byte locations in the memory map. The byte ordering is "big endian" as shown in Figure 55. However, no assumptions are made about the order in which bytes are written into multi-byte registers.

Unused bits in the memory map will return a 0 when read except for unused bits in registers holding signed integers. In this case, the most significant bit of the register will be sign extended. For example, a 12 bit signed register will be sign extended to fill a 16 bit memory map location (two bytes). A 16 bit memory map location holding a 12 bit unsigned integer will return a 0 from its most significant bits.

A.6.4.3 Reyholed address locations

In the present invention, certain less frequently accessed memory map locations have been placed behind

20

25

"keyhole". A "keyhole" has two registers associated with it, a keyhole address register and a keyhole data register.

The keyhole address specifies a location within an extended address space. A read or a write operation to the keyhole data register accesses the location specified by the keyhole address register.

After accessing a keyhole data register the associated keyhole address register increments. Random access within the extended address space is only possible by writing a new value to the keyhole address register for each access.

A chip in accordance with the present invention, may have more than one "keyholed" memory map. There is no interaction between the different keyholes.

A.6.5 Special registers

15 A.6.5.1 Unused registers

Registers or bits described as "not used" are locations in the memory map that have not been used in the current implementation of the device. In general, the value 0 can be read from these locations. Writing 0 to these locations will have no effect.

As will be appreciated by one of ordinary skill in the art, in order to maintain compatibility with future variants of these products, it is recommended that the user's software should not depend upon values read from the unused locations. Similarly, when configuring the device, these locations should either be avoided or set to the value 0.

A.6.5.2 Reserved registers

Similarly, registers or bits described as "reserved" in the present invention have un-documented effects on the behavior of the device and should not be accessed.

A.6.5.3 Test registers

Furthermore, registers or bits described as "test registers" control various aspects of the device's

EL If

testability. Therefore, these registers have no application in the normal use of the devices and need not be accessed by normal device configuration and control software.

SECTION A.7 Clocks

In accordance with the present inventions, many different clocks can be identified in the video decoder system. Examples of clocks are illustrated in Figure 56.

As data passes between different clock regimes within the video decoder chip-set, it is resynchronized (on-chip) to each new clock. In the present invention, the maximum frequency of any input clock is 30 MHz. However, one of ordinary skill in the art will appreciate that other frequencies, including those greater than 30MHz, may also be used. On each chip, the microprocessor interface (MPI) operates asynchronously to the chip clocks. In addition, the Image Formatter can generate a low frequency audio clock which is synchronous to the decoded video's picture rate. Accordingly, this clock can be used to provide audio/video synchronization.

A.7.1 Spatial Decoder clock signals

The Spatial Decoder has two different (and potentially asynchronous) clock inputs:

Signal Name	input /	Description
coded_clock	input	This clock controls data transfer in to the coded data
		port of the Spatial Decoder.
		On-chip this clock controls the processing of the
		coded data until it reaches the coded data buffer.
decoder_clock input		The decoder clock controls the majority of the
		processing functions on the Spatial Decoder.
		The decoder clock also controls the transfer of data
		out of the Spatial Decoder through its output port.

Table A.7.1 Spatial Decoder clocks

20

5

10



A.7.2 -Temporal Decoder clock signals

The Temporal Decoder has only one clock input:

Signal Name	input / Output	Description
decoder_clock	Input	The decoder clock controls all of the processing
		functions on the Temporal Decoder.
		The decoder clock also controls transfer of data in to
# 1		the Temporal Decoder through its input port and out
i		via its output port.

Table A.7.2 Temporal Decoder clocks

A.7.3 Electrical specifications

Num.	Characteristic	30 MHz			
Non.	Characteristic	Min,	Max.	Unit	Note
35	Clock period	33		กร	
36	Clock high period	13		ns	
37	Clock low period	13		ns	

Table A.7.3 Input clock requirements

, ,

Symbol	Parameter	Min.	Max.	Units
V _{IH}	Input logic '1' voltage	3.68	V ₂₀ + 0.5	٧
٧٠ر	Input logic '0' voltage	GND - 0.5	1,43	٧
loz	input leakage current		± 10	μΑ

Table A.7.4 Clock input conditions

A.7.3.1 CMOS levels

The clock input signals are CMOS inputs. V_{IHmin} is approx. 70% of V_{DD} and V_{ILmax} is approx. 30% of V_{DD} . The values shown in Table A.7.4 are those for V_{IH} and V_{IL} at their respective worst case V_{DD} . $V_{DD} = 5.0 \pm 0.25 V$.

A.7.3.2 Stability of clocks

In the present invention, clocks used to drive the DRAM interface and the chip-to-chip interfaces are derived from the input clock signals. The timing specifications for these interfaces assume that the input clock timing is stable to within \pm 100 ps.

ni in nii 1

15

20

25

30

SECTION A.8 JTAG

As circuit boards become more densely populated, it is increasingly difficult to verify the connections between components by traditional means, such as in-circuit testing using a bed-of-nails approach. In an attempt to resolve the access problem and standardize on a methodology, the Joint Test Action Group (JTAG) was formed. The work of this group culminated in the "Standard Test Access Port and Boundary Scan Architecture", now adopted by the IEEE as standard 1149.1. The Spatial Decoder and Temporal Decoder comply with this standard.

The standard utilizes a boundary scan chain which serially connects each digital signal pin on the device. The test circuitry is transparent in normal operation, but in test mode the boundary scan chain allows test patterns to be shifted in, and applied to the pins of the device. The resultant signals appearing on the circuit board at the inputs to the JTAG device, may be scanned out and checked by relatively simple test equipment. By this means, the inter-component connections can be tested, as can areas of logic on the circuit board.

All JTAG operations are performed via the Test Access Port (TAP), which consists of five pins. The Trst (Test Reset) pin resets the JTAG circuitry, to ensure that the device doesn't power-up in test mode. The tck (Test Clock) pin is used to clock serial test patterns into the tdi (Test Data Input) pin, and out of the tdo (Test Data Output) pin. Lastly, the operational mode of the JTAG circuitry is set by clocking the appropriate sequence of bits into the tms (Test Mode Select) pin.

The JTAG standard is extensible to provide for additional features at the discretion of the chip manufacturer. On the Spatial Decoder and Temporal Decoder,

there are 9 user instructions, including three JTAG mandatory instructions. The extra instructions allow a degree of internal device testing to be performed, and provide additional external test flexibility. For example, all device outputs may be made to float by a simple JTAG sequence.

For full details of the facilities available and instructions on how to use the JTAG port, refer to the following JTAG Applications Notes.

10 A.8.1 Connection of JTAG pins in non-JTAG systems

Signal	Direction	Description
trst	input	This pin has an internal pull-up, but must be taken
		low at power-up even if the JTAG features are not
		being used. This may be achieved by connecting
<u> </u>		trst in common with the chip reset pin reset.
tdi	Input	These pins have internal pull-ups, and may be left
tms		disconnected if the JTAG circuitry is not being used
tck	Input	This pin does not have a pull-up, and should be tied
		to ground if the JTAG circuitry is not used.
100	Спри	High impedance except during JTAG scan
		operations. If JTAG is not being used, this pin may
		be left disconnected.

Table A.8.1 How to connect JTAG inputs

A.8.2 -Level of Conformance to IEEE 1149.1

A.8.2.1 Rules

All rules are adhered to, although the following should be noted:

Rules	Description	
3.1.1(b)	The trst pin is provided.	
3.5.1(b)	Guaranteed for all public instructions (see IEEE 1149.1	
	5.2.1(c)).	
5.2.1(c)	Guaranteed for all public instructions. For some private	
	instructions, the TDO pin may be active during any of the	
	States Capture-DA, Exit1-DA, Exit-2-DA & Pause-DA.	
5.3.1(a)	Power on-reset is achieved by use of the trst pin.	
6.2.1(e,f)	A code for the BYPASS instruction is loaded in the Test-Logic-	
	Reset state.	
7.1.1(d)	Un-allocated instruction codes are equivalent to BYPASS.	
7.2.1(c)	There is no device ID register.	

Table A.8.2 JTAG Rules

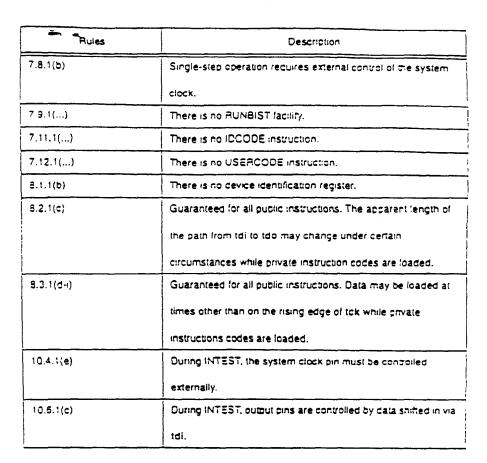


Table A.S.2 JTAG Rules

A.8.2.2 Recommendations

Recommendation	Oescription	
3.2.1(b)	tck is a high-impedance CMOS input.	
3.3.1(c)	tms has a high impedance pull-up.	
3.6.1(d)	(Applies to use of chip).	
3.7.1(a)	(Applies to use of chip).	
6.1.1(e)	The SAMPLE/PRELOAD instruction code is loaded during	
	Capture-IR.	
7.2.1(1)	The INTEST instruction is supported.	
7.7.1(g)	Zeros are loaded at system output pins during EXTEST.	
7.7.2(h)	All system outputs may be set high-impedance.	
7.8.1(1)	Zeros are loaded at system input pins during INTEST.	
8.1 1(d.e)	Design-specific test data registers are not publicly accessible.	

Table A.8.3 Recommendations met

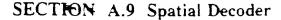
Recommendation	Description
10.4.1(1)	During EXTEST, the signal driven into the on-chip logic from
	the system clock pin is that supplied externally.

Table A.8.4 Recommendations not implemented

A.8.2.3 Permissions

Permissions	Description	
3.2.1(c)	Guaranteed for all public instructions.	
6.1.1(1)	The instruction register is not used to capture design-specific	
	information.	
7.2.1(g)	Several additional public instructions are provided.	
7.3.1(a)	Several private instruction codes are allocated.	
7.3.1(c)	(Rule?) Such instructions codes are documented.	
7.4.1(1)	Additional codes perform identically to BYPASS.	
10.1.1(i)	Each output pin has its own 3-state control.	
10.3.1(h)	A parallel latch is provided.	
10.3.1(i,j)	During EXTEST, input pins are controlled by data shifted in via	
	tdf.	
10.5.1(d,e)	3-state cells are not forced inactive in the Test-Logic-Reset	
	state.	

Table A.8.5 Permissions met



- · 30 MH, operation
- Decodes MPEG, JPEG & H.261
- · Coded data rates to 25 Mb/s
- 5 Video data rates to 21 MB/s
 - ·Flexible chroma sampling formats
 - · Full JPEG baseline decoding
 - · Glue-less DRAM interface
 - ·Single -5V supply
- 10 · 208 pin PQFP package
 - Max. power dissipation 2.5W
 - Independent coded data and decoder clocks
 - ·Uses standard page mode DRAM

The Spatial Decoder is a configurable VLSI decoder chip for use in a variety of JPEG, MPEG and H.261 picture and video decoding applications.

In a minimum configuration, with no off-chip DRAM, the Spatial Decoder is a single chip, high speed JPEG decoder. Adding DRAM allows the Spatial Decoder to decode JPEG encoded video pictures. 720x480, 30Hz, 4:2:2 "JPEG video" can be decoded in real-time.

With the Temporal Decoder Temporal Decoder the Spatial Decoder can be used to decode H.261 and MPEG (as well as JPEG). 704x480, 30Hz, 4:2:0 MPEG video can be decoded.

Again, the above values are merely illustrative, by way of example and not necessarily by way of limitation, of typical values for one embodiment in accordance with the present invention. Accordingly, those of ordinary skill in the art will appreciate that other values and/or ranges may be used.

A.9.1 Spatial Decoder Signals

, Siçnal Name	vo	Pin Number	Description
coded_clock	1	182	Coded Data Port, Used to supply
coded_data(7:0)	1	172, 171, 169, 168, 167, 166, 164,	coded data or Tokens to the Spatial
		163	Decoder.
coded_extn	1	174	
coded_valid	1	162	See sections A.10.1 and
coded_accept	0	161	A.4.1
byle_mode	1	176	
enable(1:0)	1	126, 127	Micro Processor Interface (MPI).
~	1	125	
addr(6:0)	1	136, 135, 133, 132, 131, 130, 128	
data[7:0]	0	152, 151, 149, 147, 145, 143, 141,	See section A.6.1
		140	
īrq	0	154	
DRAM_data(31:0)	VO	15, 17, 19, 20, 22, 25, 27, 30, 31,	DRAM interface.
		33, 35, 38, 39, 42, 44, 47, 49, 57,	
		59, 61, 63, 66, 68, 70, 72, 74, 76,	See section A.5.2
		79, 81, 83, 84, 85	
CRAM_addr(10:0)	0	184, 186, 188, 189, 192, 193, 195.	
ł		197, 199, 200, 203	
RAS	0	11	
CAS(3:0)	0	2, 4, 6, 8	
WE	0	12	
ŌĒ	0	204	
DRAM_enable	1	112	
out_data(8:0]	0	88, 89, 90, 92, 93, 94, 95, 97, 98	Output Port.
out_extn	0	87 .	See section A.4.1
_cut_valid	0	99	
, out_accept	1	100	
tcx	1	115	JTAG port
tei	1	116	See section A.B
tco	0	120	
tms	1	117	
trst	1	121	

Table A.9.1 Spatial Decoder signals

Signal Name	5	Pin Number	Description
decoder_clock	1	177	The main decoder clock. See section
			A.7
reset	1	160	Reset

Table A.9.1 Spatial Decoder signals (contd)

Signal Name	NO	Pin Num.	Description	
tph0ish	ı	122	If override = 1 then tph0ish and tph1ish are	
tph1ish	1	123	inputs for the on-chip two phase clock.	
override	1	110	For normal operation set override = 0, tph0ish and tph1ish are ignored (so connect	
chiptest	1	111	to GND or V _{DD}). Set chiptest = 0 for normal operation.	
tloop	1	114	Connect to GND or V _{DD} during normal	
			operation.	
ramtest	1	109	If ramtest = 1 test of the on-chip RAMs is enabled. Set ramtest = 0 for normal operation.	
pliselect	**	178	If pliselect = 0 the on-chip phase locked loops are disabled. Set pliselect = 1 for normal operation.	
ti	1	180	Two clocks required by the DRAM interface	
tq	1	179	during test operation.	
			Connect to GND or V _{DD} duing normal operation.	
pdout		207		
pdin		206	These two pins are connections for an external filter for the phase lock loop.	

Table A.9.2 Spatial Decoder Test signals



Signal Name	Pig	Signal Name	P	n Signal Name	P	n Signal Name	
nc	208	ne	156	nc	104		
test oin	207	nc	155	nc	103		5
test oin	206	īrq	154	nc	102		
GND	205	nc	153	VDO	101	DRAM_data[15]	5
OE	204	data[7]	152	out_accept	1100		- 4
DRAM_addr(0)	203	data(6)	151	out_valid	99	inc inc	148
VDD	202	nc	150	out_data[0]	98	DRAM_data(16)	4
ne	201	data(5)	149	out_data[1]	97	nc	46
DRAM_acdr(1)	200	ne	148	GND	96	GND	145
DRAM_addr(2)	199	data(4)	147	out_data(2)		DRAM_cata[17]	
GND	198	GND	146	out_data(3)	95	inc inc	43
DRAM_addr(3)	197	data(3)	145	out_data[4]	94	DRAM_data(18)	42
nc	196	ne	144	out_data(5)	93	VDD	4:
DRAM_addr(4)	195	da!a[2]	143	VDO	92	Inc	140
VDD	194	ne	142	out_data[6]	91	DRAM_data(19)	39
CRAM_addr(5)	193	data(1)	141	out_data[7]	90	DRAM_data(20)	38
DRAM_addr(6)	192	data(0)	140		89	nc	37
nc		nc	139	out_data(8)	88	GND	36
3NO		VDD	138	GND	87	DRAM_data(21)	135
CRAM_addr[7]	1	nc	137	DRAM_data[0]	86	nc	134
PAM_acdr(8)		addr[6]	136	DRAM_data[1]	85	DRAM_data(22)	j33
'DO	1	addr(5)	135	DRAM_data[2]	84	VDD	32
RAM_addr(9)	; 	GND	134	VDD	83	DRAM_data(23)	31
c	185	addr[4]	133	DRAM_data(3)	82	DRAM_data[24]	30
PAM_addr(10)		eddr(3)	132	nc Cata(3)	81	Inc Inc	29
IND		addr[2]	131	DRAM_data(4)	80	GND	25
oded_ctock	+	addr(1)	130	GND	79	DRAM_cata(25)	27
DO	+ +	VDD	129	nc	78	nc	25
est pin	 	addr(0)		DRAM_data[5]	77	DRAM_data(25)	25
est pin	1 1-	nable[0]	127	nc nc	! !	nc VDD	24
est pin		nable(1)		DRAM_data(6)	1 1	VDD	23
ecoder_clock	-	~	<u> </u>	ADD DEST(0)	: 	ORAM_data(27)	122
yte_mode	; 	and		DRAM_data[7]	1 1	00.444	:21
ND	1	est pin	1	DHAM_02(2(/)	1	ORAM_data(28)	120
oded_extr	174	est pin	.23		71	ORAM_data[29]	19

Table A.9.3 Spatial Decoder Pin Assignments



						226				
Signal Name		Pin	Signal Name		Pin	Signal Name		ع. _۲	Signal Name	
ne		80	nc	1	56	пс		104	ne	
lest pin	2	07 1	nc	1	55	ne		103		
lest pin	2	06 i	rq	1	54	ne		:C2	l ne	
GND	2	05 n	nc .	1	53	V00		101	nc	
OE	20	04 e	ata[7]	1	52	out_accept			DRAM_data(15)	
DRAM_addr(0)	20)3 d	ata(6)	19	51	out_vand		:00	re	4
VDD	20)2 n	c		50	out_cata(0)		9	OFAM_cata(15)	4
пс	20	1 d	ata(5)	14				а	ac .	4
DRAM_addr(1)	20			14		Out_data(1)	9	7	GND	49
DRAM_addr(2)	19	+-	1(2(4)		-+	GND	9(5	DRAM_data(17)	4.4
GND	198		ND ND	14	+	out_data(2)	95	5	nc	43
DRAM_addr(3)	197	-+	ta[3]	144	-	out_data(3)	94	•	DRAM_data(18)	42
nc	196	-		14!	+	ou!_data(4)	93		YDD	41
DRAM_addr[4]	195	+-		144	+	out_data(5)	92		ne	40
VDD	194	-	(2)	143	+	VDD	91	T	DRAM_data(19)	39
DRAM_addr[5]		+		142	1	ou_data[6]	90		DRAM_data[20]	38
DRAM_addr[6]	193	+	a(1)	141	1	out_data[7]	89	1	ne	37
nc	192	+	a (0)	140	1	out_data(8)	88	0	IND	35
GND	191	nc		139	٥	out_extn	87	0	RAM_data(21)	35
DPAM_addr(7)	190	VDI) 	138	9	SND	86	n		34
DRAM_addr(8)	189	nc		137	0	PAM_data(0)	85	0	RAM_data(22)	33
ADD Secure	188	addi	[6]	136	O	PAM_data(1)	8-4	+-	00	32
	187	addr	(5)	135	O	RAM_data(2)	83	01	PAM_data(23)	31
DRAM_addr[9]	186	GNE		134	VI	00	82	÷-	RAM_data(24)	30
nc no	185	addr	[4]	133	Df	RAM_data[3]	81	nc		29
ORAM_addr[10]	184	Poppe	3)	132	nc	:	80	GA		
GND	183	addr	2]	131	DF	RAM_data(4)	79	[AM_data(25)	28
oded_clock	182	addr(1]	130	G٨		78	nc		27
/00	181	ססע		129	nc		77		AM_data(25)	25
est pin	180	addr(C	oj	125	DA	IAM_data(5)	76	nc		25
St pin	179	enable	(0)	127	nc		75	VD	<u> </u>	24
est pin	178	enable	(1)	126	DR	AM_data[6]	74			23
ecoder_clock	177	r w		125	VD	·	73		W_data[27]	22
yle_mode	176	GND		124	DRA	AM_data(7)		nc		21
ND	175	lest pir	1		nc	(-)	72		M_data[29]	20
ded_extri	174	lest pin				LM_data[8]	71	CPA ——	M_data[29]	٠.٠

Table A.9.3 Spatial Decoder Pin Assignments

Signal Name	Pin	Signal Name	Pin	Signal Name	2·n	Signal Name	Pin
rc	173	trai	121	GND	69	DPAM_data(30)	17
coded_data[7]	172	tdo	120	DPAM_data(9)	68	nc	15
coded_data(6)	171	nc	119	nc	57	CRAM_data[31]	15
V00	170	VD0	118	DRAM_data(10)	66	VD0	14
coded_data(5)	159	trus	117	VDO	65	ne	13
coded_data[4]	168	tdi	116	nc	54	WE	12
coded_data[3]	167	tck	115	DRAM_data[11]	53	FAS	11
coded_data(2)	166	test pin	114,	ne	52	nc	10
GND	165	GND	113	DRAM_data(12)	61	GND	9
coded_data(1)	164	DFAM_enable	112	GND	60	CAS(0)	3
coded_data(0)	163	test pin	111	DFAM_data(13)	59	nc	17
coded_vaiid	152	test pin	110	ne	58	CAS(I)	; 6
coded_acces(161	test pin	109	DRAM_data[14]	57	VDO	15
reset	160	nc	108	VDD	; 55	CAS(2)	4
VDD	159	nc	107	nc	55	nc	3
ne	158	nc	106	ne	54	CAS(3)	2
лс	157	nc	105	ne	53	nc	1

Table A.9.3 Spatial Decoder Pin Assignments (contd)

A.9.1.1 "nc" no connect pins

The pins labeled nc in Table A.9.3 are not currently used these pins should be left unconnected.

5 A.9.1.2 V_{DD} and GND pins

As will be appreciated by one of ordinary skill in the art, all the $V_{\rm DD}$ and GND pins provided should be connected to the appropriate power supply. Correct device operation

cannot be ensured unless all the $V_{\rm DD}$ and GND pins are correctly used.

A.9.1.3 Test pin connections for normal operation

Nine pins on the Spatial Decoder are reserved for internal test use.

Pin number	Connection
	Connect to GND for normal operation
; ;	Connect to V ₅₀ for normal operation
i	Leave Open Circuit for mormal operation

Table A.9.4 Default test pin connections

A.9.1.4 JTAG pins for normal operation See section A.8.1.



A.9.2 - Spatial Decoder memory map

Addr. (hex)	Register Name	See table
0x00 0x03	Interrupt service area	A.9.6
0x04 0x07	Input circuit registers	A.9.7
0x08 0x0F	Start code detector registers	
0x10 0x15	Buffer start-up control registers	A.9.8
0x16 0x17	Not used	
0x18 0x23	DRAM interface configuration registers	A.9.9
0:24 0:26	Buffer manager access and keyhole registers	A.9.10
0x27	Not used	
0x28 0x2F	Huffman decoder registers	A.9.13
0x30 0x39	Inverse quantiser registers	A.9.14
0x3A 0x3B	Not used	
0x3C	Reserved	
0x3D 0x3F	Not used	
0x40 0x7F	Test registers	

Table A.9.5 Overview of Spatial Decoder memory map





	Add	s. Br		
	(hex) num	Register Name	Page references
	0x00	7	chip_event CED_EVENT_0	
		6	not used	
	İ	5	illegal_length_count_event	
	<u> </u>		SCD_ILLEGAL_LENGTH_COUNT	
		4	reserved may read 1 or 0	
			SCO_JPEG_OVERLAPPING_START	
		3	overlapping_start_event	
			SCD_NON_JPEG_OVERLAPPING_START	
		2	unrecognised_start_event	
			SCD_UNRECOGNISED_START	
		1	stop_after_picture_event	
			SCD_STOP_AFTER_PICTURE	
		0	non_aligned_start_event	
			SCD_NON_ALIGNED_START	
Ţ	0x01	7	chip_mask CED_MASK_0	
İ		6	not used	
:		5	illegal_length_count_mask	
!		4	reserved write 0 to this location	
			SCD_JPEG_OVERLAPPING_START	
		3	non_jpeg_cverlapping_start_mask	
:		2	unrecognised_start_mask	
!		1	stop_after_picture_mask	
L		0	non_aligned_start_mask	
	0x02	7	idct_too_lew_event IDCT_DEFF_NUM	
		6	idct_too_many_event IDCT_SUPER_NUM	
		5	accept_enable_event BS_STREAM_END_EVENT	
		4	target_met_event BS_TARGET_MET_EVENT	
		3	counter_flushed_too_early_event	
:			BS_FLUSH_BEFORE_TARGET_MET_EVENT	
		2	counter_flushed_event BS_FLUSH_EVENT	
	Ì	t	parser_event DEMUX_EVENT	
		0	hutiman_event HUFFMAN_EVENT	
			able A C C Interrupt and in	

Table A.9.6 Interrupt service area registers

Addr.	Sa		
(hex)	num.	Register Name	Page references
0x03	7	idct_too_few_mask	
	6	idct_too_many_mask	
	5	accept_enable_mask	
	4	target_met_mask	
	3	counter_flushed_too_early_mask	
	2	counter_flushed_mask	
	. 1	parser_mask	
	0	huffman_mask	

Table A.9.6 Interrupt service area registers (contd)

Adar.	Brt	Secion 4	
(hex)	ոսт.	Register Name	Page references
0x04	7	coded_busy	
	6	enable_mpl_input	
	5	coded_extn	
	4:0	not used	
Cx05	7:0	coded_data	
0x06	7:0	not used	
0x07	7:0	not used	
80x0	7:1	not used	
	0	start_code_detector_access	
		also input_circuit_access	
		CED_SCD_ACCESS	
0x09	7:4	not used CED_SCD_CONTROL	
	3	stop_after_picture	
	2	discard_extension_data	
	1	discard_user_data	
	0	ignore_non_aligned	
0x0A	7:5	not used CED_SCD_STATUS	
	4	insert_sequence_start .	
	3	discard_all_data	
	2:0	start_code_search	
-	Table A 6	7.04	

Table A.9.7 Start code detector and input circuit registers

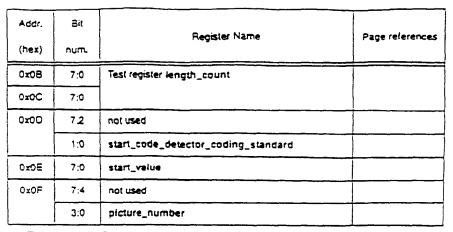


Table A.9.7 Start code detector and input circuit registers (contd)

Addr.	Bit	Resister Name	See ale
(hex)	חטרת.	Register Name	Page references
0x10	7:1	not used ·	
	0	startup_access CED_BS_ACCESS	
0x11	7:3	not used	
	2:0	bit_count_prescale CED_BS_PRESCALE	
0x12	7:0	bit_count_target CED_BS_TARGET	
0x13	7;0	bit_count CED_BS_COUNT	
10414	7:1	not used	
	0	offchip_queue CED_BS_OUEUE	
0x15	7:1	not used	
	0	enable_stream CED_BS_ENABLE_NXT_STM	

Table A.9.8 Buffer start-up registers

Addr.	Bit		
(hex)	ոսт.	Register Name	Page references
0x18	7:5	not used	
	4:0	page_start_length	
		CED_IT_PAGE_START_LENGTH	
0x19	7:4	not used	
	3.0	read_cycle_length	
OxIA	7:4	not used	
	3:0	write_cycle_length	

Table A.9.9 DRAM interface configuration registers

An	Addr. Si			
	(hex) num.		Register Name	Page references
0x1	8 =	7:4	not used	
<u> </u>		3:0	refresh_cycle_length	
0x1	c L	7:4	not used	
		3:0	CAS_falling	
0x10		7:4	not used	
		3:0	RAS_falling	· — · — — — — — — — — — — — — — — — — —
0x15		7:1	not used	
		0	interface_timing_access	
0x1F		7:0	refresh_interval	
0x20		7	not used	
		5:4	DRAM_addr_strength(2:0)	
		3:1	CAS_strength[2:0]	
		0	RAS_strength[2]	
0x21	7	:5	RAS_strength[1:0]	
	5	:3	OEWE_strength(2:0)	
	2	:0	DRAM_data_strength(2:0)	
Cx22	7		ACCESS bit for pad strength etc. ?not	
		f	ISMCED_DRAM_CONFIGURE	
	6	. 1	tero_buffers	
	5	2	DRAM_enable	
	4	п	o_refresh	
	3:2	2 10	pw_address_bits(1:0)	
	1:0		RAM_data_width(1:0)	
0×23	7:0		est registers CED_PLL_RES_CONFIG	
7.			2	

Table A.9.9 DRAM interface configuration registers (contd)

Addr.	Вп					
(hex)	חטת	Register Name	Page references			
0x24	7:1	not used				
	0	buffer_manager_access				
0x25	7:6	not used				
	5:0	buffer_manager_keyhole_address				
0x26	7:0	buffer_manager_keyhole_data				
Table A 9 10 Buffer man						

Table A.9.10 Buffer manager access and keyhole registers

		250	
Addr.	Brt		
(hex)	num	Register Name	Page references
0x00	7:0	not used	
0x01	72		
	1:0	cdb_base	
0x02	7:0		
0x03	7:0	1	
0x04	7:0	not used	
0x05	7:2		
	1:0	cdb_length	
0x06	7:0		
0×07	7:0		
80x0	7:0	not used	
0×09	7:0	cdb_read	
A0x0	7:0		
0x08	7:0		
0x0C	7:0	not used	
0×0D	7:0	cdb_number	
0x0E	7:0		and the second s
0x0F	7:0		
0×10	7:0	not used	
0x11	7:0	tb_base	
0x12	7:0		1
0x13	7:0		
0x14	7:0	not used	
0x15	7:0	tb_length	
Cx 16	7:0		
0x17	7:0		
0x18	7:0	not used	
· 0x19	7:0	tb_read	
0x1A	7:0		
0x1B	7:0		
0x1C	7:0	not used	
0x1D	7:0	tb_number	
0x1E	7:0		
0x1F	7:0		
	'	C 44 Duffer	· - · · · · · · · · · · · · · · · · · ·

Table A.9.11 Buffer manager extended address space

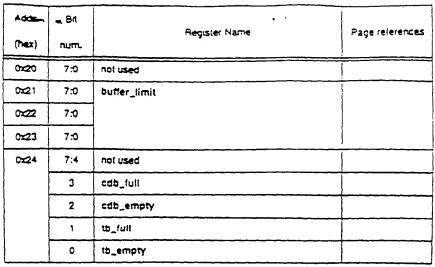


Table A.9.11 Buffer manager extended address space (contd)

Addr.	ЯB	Register Name	Page references
(hex)	יחונוי	They were the the	1 age references
0×28	7	demux_access CEO_H_CTRL[7]	
	6:4	huffman_error_code(2:0) CED_H_CTRL(6:4)	
	3:0	private huttman control bits [3] selects special	
		CBP, [2] selects 4/8 bit fixed length CBP	
0×29	7:0	parser_error_code CED_H_DMUX_ERR	
0x2A	7:4	not used	
	3:0	demux_keyhole_address	
0×28	7:0	CED_H_KEYHOLE_ADDR	
0±2C	7:0	demux_keyhole_data CED_H_KEYHOLE	
0×20	7	dummy_last_picture CED_H_ALU_REGO,	
		r_dummy_last_frame_bit	
	6	field_into CED_H_ALU_REGO, r_field_into_bit	
	5:1	not used	
	0	continue CED_H_ALU_REGO, r_continue_bit	
0x2E	7:0	rom_revision CED_H_ALU_REG1	
0x2F	7:0	private register	

Table A.9.12 Video demux registers

Addr.	Вп		
(hex)	num.	Register Name	Page reference:
0x2F	7	CED_H_TRACE_EVENT write 1 to single step, one	
		will be read when the step has been completed	
	6	CED_H_TRACE_MASK set to one to enter single	
		step mode	
	5	CED_H_TRACE_RST partial reset when sequenced	
L		1,0	
Γ	4:0	not used	

Table A.9.12 Video demux registers (contd)

			
Addr.	Bit		
(hex)	num.	Register Name	Page references
C×00	7:0	not used	
0x0F			
0x10	7:0	horiz_pels r_honz_pels	
0x11	7:0		
0x12	7:0	vert_pels r_vert_pels	
0x13	7:0		
Cx14	7:2	not used	
	1:0	buffer_size r_buffer_size	
Cx15	7:0		
0x16	7:4	not used	
	3:0	pel_aspect r_pel_aspect	
9x17	72	not used	
	1:0	bit_rate r_bit_rate	
Cx18	7:0		
0x19	7:0		
Cx1A	7:4	not used	
	3:0	pic_rate r_pic_rate	
0x1B	7:1	not used	
	0	constrained /_constrained	
Gx1C	7:0	picture_type	
0x:D	7:0	h261_pic_type	

Table A.9.13 Video demux extended address space (Sheet 1 of 8)

A Discontinuity of the control of th



	1		
Addr.	Bd	Register Name	Page references
0x1E	72	not used	
	1:0	broken_closed	
0x1F	7:5	not used	
	4:0	prediction_mode	
0x20	7:0	vbv_delay	
0x21	7:0		
0x22	7:0	private register MPEG full_pel_fwd, JPEG	
		pending_frame_change	
0x23	7:0	private register MPEG full_pel_bwd, JPEG	
		restart_index	
0x24	7:0	private register horiz_mb_copy	
0x25	7:0	pic_number	
0x26	7:1	not used	
	1:0	max_h	
0x27	7:1	not used	
	1:0	max_v	
0×28	7:0	private register scratch1	
0×39	7:0	private register scratch2	
0x2A	7:0	private register scratch3	
0 ×2 8	7:0	Nf MPEG unused1, H251 ingob	
0×2C	7:0	private register MPEG first_group, JPEG first_scan	
020	7:0	private register MPEG in_picture	
0:25	7	dummy_last_picture r_rom_control	
	6	field_info	
	5:1	not used	
	0	continue	
0 × 2F	7:0	rom_revision	
0x30	72	not used	
	1:0	dc_huff_0	
0x31	7:2	not used	
	1:0	dc_huff_1	
0x32	7:2	not used	7 " ! •
	1:0	dc_huff_2	
		· · · · · · · · · · · · · · · · · · ·	

Table A.9.13 Video demux extended address space (Sheet 2 of 8)



	Bri		
Addr.	0.0		
(hex)	กบก	Register Name	Page references
0×33	72	not used	
	1:0	dc_huff_3	
0x34	72	not used	
	1:0	ac_huff_0	
0x35	72	not used	
	1:0	ac_huff_1	
0x36	7:2	not used	
	1:0	ac_huff_2	
0×37	7:2	not used	
<u> </u>	1:0	ac_huff_3	
0x38	7:2	not used	
	1:0	10010	
0×39	7:2	not used	
	1:0	tq_1 r_tq_1	
0x3A	7:2	not used	
	1:0	tq_2 r_tq_2	
0x38	7:2	not used	
	1:0	tq_3 r_tq_3	
0x3C	7:0	component_name_0 r_c_0	
0×3D	7:0	component_name_1 r_c_1	
0x3E	7:0	component_name_2 /_c_2	
0x3F	7:0	component_name_3 r_c_3	
0x40	7:0	private registers	
0x63			
0x40	7:0	r_dc_pred_0	
0x41	7:0		
0x42	7:0	r_dc_pred_1	
0x43	7:0		
0x44	7:0	r_dc_pred_2	
0x45	7:0		
0x46	7:0	r_dc_pred_3	
	7:0		
	7:0	not used	
0x4F			

Table A.9.13 Video demux extended address space (Sheet 3 of 8)

в и



(nex) num. Register Name Page references	Addr.	Br		
0x52	(hex)	חטות.	Register Næme	Page references
0x52 7:0 r_prev_mvl 0x54 7:0 r_prev_mvb 0x55 7:0 r_prev_mvb 0x56 7:0 r_prev_mvb 0x57 7:0 r_prev_mvb 0x57 7:0 r_prev_mvb 0x58 7:0 not used 0x57 7:0 r_horiz_mbcnt 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 veri_macroblocks r_vert_mbs 0x66 7:0 veri_macroblocks r_vert_mbs 0x67 7:0 prvate register r_estart_ent 0x68 7:0 prvate register r_estart_ent 0x69 7:0 prvate register r_blk_h_cnt 0x60 7:0 prvate register r_blk_y_cnt 0x60 7:0 prvate register r_blk_y_cnt 0x60 7:0 prvate register r_blk_y_cnt 0x70 7:0 coding_standard r_coding_std 0x71 7:0 </td <td>0x50</td> <td>7:0</td> <td>r_prev_mhf</td> <td> </td>	0x50	7:0	r_prev_mhf	
Cx53 7:0 r_prev_mhb 0x55 7:0 r_prev_mhb 0x56 7:0 r_prev_mvb 0x57 7:0	Cx51	7:0	_	
0x54 7:0 r_prev_mhb 0x55 7:0 r_prev_mwb 0x57 7:0 0 0x57 7:0 not used 0x58 7:0 not used 0x55 7:0 not used 0x60 7:0 r_horiz_mbcnt 0x63 7:0 noriz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x66 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 vert_macroblocks r_vert_mbs 0x68 7:0 prvate register r_restart_ent 0x69 7:0 prvate register r_restart_ent 0x69 7:0 private register r_blk_n_cnt 0x60 7:0 private register r_blk_n_cnt 0x60 7:0 prvate register r_blk_n_cnt 0x60 7:0 prvate register r_compid 0x60 7:0 prvate register r_compid 0x70 7:0 prvate register r_brdr_r_size 0x71	0x52	7:0	r_prev_mvt	1
0x55 7:0 r_prev_mvb 0x57 7:0 r_prev_mvb 0x57 7:0 not used 0x5F 7:0 not used 0x5F 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x66 7:0 provate register r_restart_cnt 0x68 7:0 provate register r_restart_int 0x69 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x60 7:0 provate register r_bk_v_cnt 0x70 7:0 provate register r_bk_v_criz_size 0x71 7:0 private register r_bwd_r_size <td>Cx53</td> <td>7:0</td> <td></td> <td></td>	Cx53	7:0		
0x56 7:0 r_prev_mvb 0x57 7:0 not used 0x5F 7:0 not used 0x5F 0x60 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x62 7:0 horiz_macroblocks r_horiz_mbs 0x63 7:0 horiz_macroblocks r_vert_mbs 0x64 7:0 vert_macroblocks r_vert_mbs 0x65 7:0 pmvale register r_restart_cnt 0x69 7:0 pmvale register r_restart_int 0x69 7:0 restart_interval r_restart_int 0x68 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_compid 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x71 7:0 private register r_blk_v_cnt 0x72 7:0	0x54	7:0	r_prev_mhb	
0x57 7:0 not used 0x5F 7:0 not used 0x5F 0x60 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x62 7:0 f_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x66 7:0 private register r_restart_cnt 0x69 7:0 private register r_restart_int 0x69 7:0 restart_interval r_restart_int 0x68 7:0 private register r_blk_h_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_compid 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x70 7:0 private register r_blk_v_cnt 0x71 7:0	0×55	7:0		
0x58 7:0 not used 0x5F 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 pmvale register r_restart_cnt 0x68 7:0 pmvale register r_restart_int 0x69 7:0 private register r_tit 0x60 7:0 private register r_blk_h_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_compid 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_blk_v_cnt 0x70 7:0 private register r_blk_v_cnt 0x71 7:0 private register r_blk_v_cnt 0x72 7:0 private register r_blk_v_cnt 0x73 7:0 private register r_blk_v_cnt 0x74 7:0 not	0x56	7:0	/_prev_mvb	
0x5F 0x60 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 pmvale register r_restart_cnt 0x68 7:0 pmvale register r_restart_int 0x69 7:0 restart_interval r_restart_int 0x68 7:0 private register r_blk_h_cnt 0x60 7:0 private register r_blk_y_cnt 0x60 7:0 private register r_compid 0x65 7:0 private register r_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_bwd_r_size 0x71 7:0 private register r_bwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used	0x57	7:0		
0x60 7:0 r_horiz_mbcnt 0x61 7:0 r_vert_mbcnt 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 0x68 0x69 7:0 private register r_restart_cnt 0x68 7:0 private register r_start_int 0x68 7:0 private register r_testart_int 0x68 7:0 private register r_blk_h_cnt 0x60 7:0 private register r_blk_v_cnt 0x60 7:0 private register r_compid 0x60 7:0 private register r_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_bwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used	0x58	7:0	not used	
0x61 7:0 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 0x68 0x68 7:0 private register r_estart_cnt 0x69 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6E 7:0 private register r_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_bwd_r_size 0x72 7:0 private register r_bwd_r_size 0x73 7:0 not used	0x5F			
0x61 7:0 0x62 7:0 r_vert_mbcnt 0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_vert_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 0x68 0x68 7:0 private register r_restart_ent 0x69 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6D 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6E 7:0 private register r_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_bwd_r_size 0x72 7:0 private register r_bwd_r_size 0x73 7:0 not used	0x60	7:0	r_horiz_mbent	
0x63 7:0 horiz_macroblocks r_horiz_mbs 0x64 7:0 horiz_macroblocks r_horiz_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 0x68 7:0 private register r_restart_ent 0x69 7:0 restart_interval r_restart_int 0x6A 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6D 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used	0x61	7:0		<u> </u>
0x64 7:0 horiz_macroblocks r_horiz_mbs 0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 private register r_restart_ent 0x68 7:0 private register r_restart_int 0x6A 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_bwd_r_size 0x72 7:0 private register r_bwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used	0x62	7:0	r_vert_mbcnt	1
0x65 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 vert_macroblocks r_vert_mbs 0x68 7:0 pmvale register r_restart_cnt 0x69 7:0 restart_interval r_restart_int 0x6A 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 pmvate register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_bwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x63	7:0		
0x66 7:0 vert_macroblocks r_vert_mbs 0x67 7:0 restart_cont 0x68 7:0 private register r_restart_int 0x6A 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cont 0x6C 7:0 private register r_blk_v_cont 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_fwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 not used	0x64	7:0	horiz_macroblocks r_horiz_mbs	1
0x67 7:0 0x68 7:0 private register r_restart_cnt 0x69 7:0 restart_interval r_restart_int 0x6A 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_hwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x55	7:0		
0x69 7:0 pmvale register r_restart_cnt 0x69 7:0 restart_interval r_restart_int 0x6B 7:0 restart_interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6D 7:0 pmvate register r_blk_v_cnt 0x6E 7:0 pmvate register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_bwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x56	7:0	vert_macroblocks r_vert_mbs	1
0x69 7:0 0x6A 7:0 restart_Interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_fwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x57	7:0		
0x6A 7:0 restart_Interval r_restart_int 0x6B 7:0 private register r_blk_h_cnt 0x6D 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_fwd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x58	7:0	private register r_restart_cnt	
0x6B 7:0 private register r_blk_h_cnt 0x6C 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x69	7:0		
0x6C 7:0 private register r_blk_h_cnt 0x6D 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x6A	7:0	restart_interval r_restart_int	<u> </u>
0x6D 7:0 private register r_blk_v_cnt 0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x6B	7:0		
0x6E 7:0 private register r_compid 0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x78 7:2 not used	0x6C	7:0	private register r_blk_h_cnt	
0x6F 7:0 max_component_id r_max_compid 0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 0x78 7:2 not used	0x6D	7:0	private register r_blk_v_cnt	
0x70 7:0 coding_standard r_coding_std 0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 not used	0x6E	7:0	private register r_compid	
0x71 7:0 private register r_pattern 0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 0x78 7:2 not used	0x6F	7:0	max_component_id r_max_compid	
0x72 7:0 private register r_twd_r_size 0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 0x78 7:2 not used	0x70	7:0	coding_standard r_coding_std	
0x73 7:0 private register r_bwd_r_size 0x74 7:0 not used 0x77 0x78 7:2 not used	0x71	7:0	private register r_pattern	
0x74 7:0 not used 0x77 0x78 7:2 not used	0x72	7:0	private register r_fwd_r_size	,
0x77	0x73	7:0	private register r_bwd_r_size	<u>'</u>
0x78 7:2 not used	0x74	7:0	not used	
	0x77			
1:0 blocks_h_0 r_blk_h_0	0x78	7:2	not used	
		1:0	blocks_h_0 r_blk_h_0	

Table A.9.13 Video demux extended address space (Sheet 4 of 8)

			243					
_	Acc	r. Br						
_	(hex	r) nun	Register Name	Page references				
	9x79	7:2	not used					
		1:0	blocks_h_1 r_blk_h_1					
	0x7	7:2	not used					
į		1:0	blocks_h_2 r_blk_h_2					
	0x78	7:2	not used					
		1:0	blocks_h_3 r_blk_h_3					
	0x7C	7:2	not used					
		1:0	blocks_v_0 r_blk_v_0					
	0x70	7:2	not used					
		1:0	blocks_v_1 r_blk_v_1					
	0x7E	7:2	not used					
		1:0	blocks_v_2 r_blk_v_2					
	0x7F	7:2	not used					
		1:0	blocks_v_3 r_bik_v_3					
	Cx7F	7:0	not used					
	OxFF							
	0x100	7:0	dc_bits_0[15:0] CED_H_KEY_DC_CP80					
	0x10F							
	0x1:0	7.0	dc_bits_1[15:0] CED_H_KEY_DC_CPB1					
	0x11F							
Ī	0x120	7:0	not used					
	0x13F							
	0x140	7:0	ac_bits_0(15:0) CED_H_KEY_AC_CPB0					
١,	0x14F		The Tark of the Ta					
\vdash	0x150	7:0	ar hite 1/16/01 CFD 11 VFD 12					
	0x15F		ac_bits_1(15:0) CED_H_KEY_AC_CPB1					
 	0x160	7:0	not used					
		7.0	not used					
-)x17F							
_	x180	7.0	dc_zssss_0 CED_H_KEY_ZSSSS_INDEX0					
	x181	7:0	dc_zssss_1 CED_H_KEY_ZSSSS_INDEX1					
	x182	7:0	not used					
—	x187							
Q:	x188	7:0	ac_eob_0 CED_H_KEY_EO8_INDEX0					
	Table		Wide a demand and a television					

Table A.9.13 Video demux extended address space (Sheet 5 of 8)



0x81F 0x820

0x82F

0x83F 0x840

0x84F

7:0

7:0

CED_KEY_MBA_CPB

CED_KEY_MVD_CPS

CED_KEY_MTYPE_I_CPS



244 Addr. Ba Register Name Page references (hex) വെന്ന. 0x189 7:0 ac_eob_1 CED_H_KEY_EOB_INDEX1 Ox18A 7:0 not used 881x0 0x18C 7:0 ac_zrt_0 CED_H_KEY_ZRL_INDEXO 0x18D 7:0 ac_zd_1 CED_H_KEY_ZAL_INDEX1 0x18E 7:0 not used 0x1FF 0x200 7:0 ac_huffval_0[161:0] CED_H_KEY_AC_fTOD_0 0x2AF 0×280 dc_huftval_0[11:0] CED_H_KEY_DC_ITOD_0 7:0 0x25F 0x2C0 7:0 not used 0×2FF 0x300 ac_huffval_1(161:0) CED_H_KEY_AC_ITOD_I 0x3AF 0x350 dc_huttval_1(11:0) CED_H_KEY_DC_ITOD_1 0x38F 0x3C0 7:0 not used 0x7FF 0x800 7:0 private registers 0xAC F 0x800 CED_KEY_TCOEFF_CP8 0x80F 0x810 CED_KEY_CBP_CPB

Table A.9.13 Video demux extended address space (Sheet 6 of 8)



	_			_	2 4 5	
	A	dar.	8	in .		
•	-	ex)	กบ	m.	Register Name	Page references
	Ox.	850	7:	0	CED_KEY_MTYPE_P_CPB	
	0x	85F				
	Cxt	360	7:0	5	CED_KEY_MTYPE_B_CPB	
	Oxe	16F				
	0x8	70	7:0		CED_KEY_MTYPE_H261_CPB	
	0×8	8F			_	:
	0x8	80	7:0	,	not used	
	0×9	00				1
	0x90)1	7:0		CED_KEY_HDSTROM_0	
Ĺ	0x90	2	7:0	0	ED_KEY_HOSTROM_1	
	0 x9 0	3	7:0	С	ED_KEY_HDSTROM_2	
	0290	F				
	0 x 91	0	7:0	ne	Dt used	
	CXAE	1				
	F					
	0xAC		7.0	CE	D_KEY_DMX_WORD_0	
<u> </u> _	0					
	CXAC		7:0	CE	D_KEY_DMX_WORD_1	
-	1					
	XAC	7	7:0	CE	D_KEY_OMX_WORD_2	
-	2	<u> </u>				
0	XAC	7	':O	CEC	D_KEY_DMX_WORD_3	
-	3	<u> </u>	_			
1	rAC	'	:0	CED	_KEY_DMX_WORD_4	
	4 AC	7:		<u> </u>		
	5	, ,.		CEU,	_KEY_DMX_WORD_5	
Cx	!	7:0	+	CED	YEV CAN INC.	
6	l	, .			KEY_DMX_WORD_6	
0x4		7:0	,	CED	KEY_DMX_WORD_7	
7						1
	$-\bot$	Α 9	13 1	مأوزن	O demuy extended add	

Table A.9.13 Video demux extended address space (Sheet 7 of 8)





Addr.	81		
(hex)	num.	Register Name	Page references
0xAC	7:0	CED_KEY_DMX_WORD_8	
8			
0xAC	7:0	CED_KEY_DMX_WORD_9	
9		-	
0xAC	7:0	not used	
A			
0xAC			
8			
0xAC	7:0	CED_KEY_DMX_AINCR	
С			
0xAC	7:0		
D			
0xAC	7:0	CED_KEY_DMX_CC	
Ε			
0xAC	7:0		
F			

Table A.9.13 Video demux extended address space (Sheet 8 of 8)

nces	
\neg	

Table A.9.14 Inverse quantiser registers

Addr.	βn					
(hex)	ոսт.	Register Name	Page references			
0x36	7:2	not used				
	1:0	lest register mpeg_indirection	İ			
0x37	7:0	not used				
0x38	7:0	iq_table_keyhole_address				
0x39	7:0	iq_table_keyhole_data				

Table A.9.14 Inverse quantizer registers (contd)

Addr.		<u> </u>
(hex)	Register Name	Page references
0x00:0x3F	JPEG Inverse quantisation table 0	
	MPEG default intra lable	
0x40:0x7F	JPEG Inverse quantisation table 1	
	MPEG default non-intra table	
0x80:0x8F	JPEG Inverse quantisation table 2	
	MPEG down-loaded intra table	
0xC0.0xFF	JPEG inverse quantisation table 3	Ţ
	MPEG down-loaded non-intra table	

Table A.9.15 Iq table extended address space

SECTION A.10 Coded data input

The system in accordance with the present invention, must know what video standard is being input for processing. Thereafter, the system can accept either pre-existing Tokens or raw byte data which is then placed into Tokens by the Start Code Detector.

Consequently, coded data and configuration Tokens can be supplied to the Spatial Decoder via two routes:

- The coded data input port
- The microprocessor interface (MPI)

The choice over which route(s) to use will depend upon the application and system environment. For example, at low data rates it might be possible to use a single microprocessor to both control the decoder chip-set and to do the system bitstream de-multiplexing. In this case, it may be possible to do the coded data input via the MPI. Alternatively, a high coded data rate might require that coded data be supplied via the coded data port.

In some applications it may be appropriate to employee a mixture of MPI and coded data port input.

A.10. - The coded data port

Signal Name	Input / Output	Description
coded_clock	Input	A clock operating at up to 30 MHz controlling the
		operation of the input circuit.
coded_data(7:0)	Input	The standard 11 wires required to implement a
coded_extn	Input	Token Port transferring 8 bit data values. See section
coded_valid	Input	A.4 for an electrical description of this
coded_accept	Output	interface.
		Circuits off-chip must package the coded data into
		Tokens.
byte_mode	input	When high this signal indicates that information is to
	Table and the same of the same	be transferred across the coded data port in byre
		mode rather than Token mode.

Table A.10.1 Coded data port signals

11

The-coded data port in accordance with the present invention, can be operated in two modes: Token mode and byte mode.

A.10.1.1 Token mode

- In the present invention, if byte_mode is low, then the coded data port operates as a Token Port in the normal way and accepts Tokens under the control of coded_valid and coded_accept. See section A.4 for details of the electrical operation of this interface.
- The signal byte_mode is sampled at the same time as data [7:0], coded_extn and coded_valid, i.e., on the rising edge of coded_clock.

A.10.1.2 Byte mode

- If, however, byte_mode is high, then a byte of data is transferred on data[7:0] under the control of the two wire interface control signals coded_valid and coded_accept. In this case, coded_extn is ignored. The bytes are subsequently assembled on-chip into DATA Tokens until the input mode is changed.
- 20 1) First word ("Head") of Token supplied in token mode.
 - 2) Last word of Token supplied (coded_extn goes low).
 - 3) First byte of data supplied in byte mode. A new DATA Token is automatically created on-chip.

A.10.2 Supplying data via the MPI

Tokens can be supplied to the Spatial decoder via the MPI by accessing the coded data input registers.

A.10.2.1 Writing Tokens via the MPI

The coded data registers of the present invention are grouped into two bytes in the memory map to allow for efficient data transfer. The 8 data bits, coded_data[7:0], are in one location and the control registers, coded_busy, enable_mpi_input and coded_extn are in a second location.

15

(Sea Table A.9.7).

When configured for Token input via the MPI, the current Token is extended with the current value of coded extn each time a value is written into coded_data[7:0]. Software is responsible for setting coded extn to 0 before the last word of any Token is written to coded data[7:0].

For example, a DATA Token is started by writing 1 into coded extn and then 0x04 into coded data[7:0]. The start of this new DATA Token then passes into the Spatial Decoder for processing.

Each time a new 8 bit value is written to coded data[7:0], the current Token is extended. Coded_extn need only be accessed again when terminating the current Token, e.g. to introduce another Token. The last word of the current Token is indicated by writing 0 to coded extn followed by writing the last word of the current Token into coded data[7:0].

Register name	SIza/Dir.	Reset State	Description
coded_extn	1	x	Tokens can be supplied to the Spatial Decoder
	rw		via the MPI by writing to these registers.
coded_data(7:0)	8	r	
	w		
coded_busy	1	1	The state of this registers indicates if the
	,		Spatial Decoder is able to accept Tokens
			written into coded_data{7:0}.
			The value 1 indicates that the interface is busy
	-		and unable to accept data. Behaviour is
			undefined if the user tries to write to
			coded_data[7:0] when coded_busy = 1
enable_mpl_input	1	0	The value in this function enable registers
	~~		controls whether coded data input to the Spatal
			Decoder is via the coded data port (0) or via the
			MPI (1).

Table A.10.2 Coded data input registers

Each time before writing to coded_data[7:0], coded_busy should be inspected to see if the interface is ready to accept more data.

A.10.3 Switching between input modes

Provided suitable precautions are observed, it is possible to dynamically change the data input mode. In general, the transfer of a Token via any one route should be completed before switching modes.

Previous mode Next Mode		Sehaviour				
Byte Token		The on-chip circuitry will use the last byte supplied in				
	MPI input	byte mode as the last byte of the DATA Token that				
1	·	it was constructing (i.e. the extribit will be set to 2"				
1		Before accepting the next Token.				

Table A.10.3 Switching data input modes

15

<u> </u>		
Previous mode	Next Mode	Benaviour
Token	: 3yte	The off-chip circuitry supplying the Taken in Taken
	• 1	mode is responsible for completing the Token is a
	•	with the extri bit of the last byte of information set to
	i i	0) before selecting byte mode.
	MPI input	Access to input via the MPI will not be granted (i.e.
		coded_busy will remain set to 1) until the off-on o
		circuitry supplying the Token in Taken mode has
		completed the Token (i.e. with the extribit of the tast
		byte of information set to 0).
MPI input	3yte	The control software must have completed the
	MPI input	Token (i.e. with the extri bit of the last byte of
		information set to 0) before enable_mpi_input is set
		to 0.

Table A.10.3 Switching data input modes (contd)

The first byte supplied in byte mode causes a DATA Token header to be generated on-chip. Any further bytes transferred in byte mode are thereafter appended to this DATA Token until the input mode changes. Recall, DATA Tokens can contain as many bits as are necessary.

The MPI register bit, coded busy, and the signal, coded_accept, indicate on which interface the Spatial decoder is willing to accept data. Correct observation of these signals ensures that no data is lost.

A.10.4 Rate of accepting coded data

In the present invention, the input circuit passes Tokens to the Start Code Detector (see section A.11). The Start code Detector analyses data in the DATA Tokens bit serially. The Detector's normal rate of

10

processing is one bit per clock cycle (of coded_clock). Accordingly, it will typically decode a byte of coded data every 8 cycles of coded_clock. However, extra processing cycles are occasionally required, e.g., when a non-DATA Token is supplied or when a start code is encountered in the coded data. When such an event occurs, the Start Code Detector will, for a short time, be unable to accept more information.

After the Start Code Detector, data passes into a first logical coded data buffer. If this buffer fills, then the Start Code Detector will be unable to accept more information.

Consequently, no more coded data (or other Tokens) will be accepted on either the coded data port, or via the MPI, while the Start Code Detector is unable to accept more information. This will be indicated by the state of the signal coded_accept and the register coded_busy.

By using coded_accept and/or coded_busy, the user is guaranteed that no coded information will be lost.

However, as will be appreciated by one of ordinary skill in the art, the system must either be able to buffer newly arriving coded data (or stop new data for arriving) if the Spatial decoder is unable to accept data.

A.10.5 Coded data clock

In accordance with the present invention, the coded data port, the input circuit and other functions in the Spatial Decoder are controlled by coded_clock. Furthermore, this clock can be asynchronous to the main decoder_clock. Data transfer is synchronized to decoder_clock on-chip.

SECTION A.11 Start code detector

A.11.1 Start codes

As is well known in the art, MPEG and H.261 coded video streams contain identifiable bit patterns called start codes. A similar function is served in JPEG by marker codes. Start/marker codes identify significant parts of the syntax of the coded data stream. The analysis of start/marker codes performed by the Start Code Detector is the first stage in parsing the coded data. The Start Code Detector is the first block on the Spatial Decoder following the input circuit.

The start/marker code patterns are designed so that they can be identified without decoding the entire bitstream.

Thus, they can be used in accordance with the present

15 invention, to help with error recovery and decoder startup. The Start Code Detector provides facilities to detect
errors in the coded data construction and to assist the
start-up of the decoder.

A.11.2 Start code detector registers

As previously discussed, many of the Start Code Detector registers are in constant use by the Start Code Detector. So, accessing these registers will be unreliable if the Start Code Detector is processing data. The user is responsible for ensuring that the Start Code Detector is halted before accessing its registers.

The register start_code_detector_access is used to halt the Start Code Detector and so allow access to its registers. The Start Code Detector will halt after it generates an interrupt.

There are further constraints on when the start code search and discard all data modes can be initiated. These are described in A.11.8 and A.11.5.1.

111

1 1 1 0 0 1 0 0 1

Register name	Size/Dk.	Reset State	Description
start_code_detector_access	1	0	Writing 1 to this register requests that the start
	rw		code detector stop to allow access to its
			registers. The user should wait until the value "
			can be read from this register indicating that
			operation has stopped and access is possible

Table A.11.1 Start code detector registers (Sheet 1 of 5)

in the state of th

Register name	Size/Dk.	Reset State	Description
illegat_length_count_event	1	0	An illegal length count event will occur if while
	~		decoding JPEG data, a length count field is
illegal_length_count_mask	1	0	found carrying a value less than 2. This should
`	~		only occur as the result of an error in the JPEG
			data.
			If the mask register is set to 1 then an interrupt
			can be generated and the start code detector
			will stop. Behaviour following an error is not
			predictable if this error is suppressed (mask
			register set to 0). See A.11.4.1
peg_overlapping_start_event	1	0	If the coding standard is JPSG and the
	~		sequence 0xFF 0xFF is found while locking for
jpeg_overlapping_start_mask	1	0	a marker code this event will occur.
	~		This sequence is a legal stuffing sequence.
			If the mask register is set to 1 then an interrupt
	į.		can be generated and the start code detector
	1		will stop, See A.11.4.2
overlapping_start_event	1	0	If the coding standard is MPEG or H.251 and
	rw.		an overlapping start code is found while looking
overlapping_start_mask	1	0	for a start code this event will occur. If the mask
	~		register is set to 1 then an interrupt can be
			generated and the start code detector will stop
			See A.11.4.2

Table A.11.1 Start code detector registers (Sheet 2 of 5)

· iji

Register name	S120/Dh.	Rosot State	Description
unrecognised_start_event	1	0	If an unrecognised start code is encountered
	~		this event will occur if the mask register is set
unrecognised_start_mask	1	0	to 1 then an interrupt can be generated and the
	rw		start code detector will stop.
start_value	8	x	The start code value read from the bitstream is
	ro		available in the register start_value while the
			start code detector is halted. See A,11,4.3
			During normal operation start_value contains
			the value of the most recently decoded starV marker code.
			Only the 4 LSBs of start_value are used during
		0	H.251 operation. The 4 MSBs will be zero. If the register stop_after_picture is set to 1
stop_after_picture_event			then a stop after picture event will be generated
stop_after_picture_mask	rw	0	after the end of a picture has passed through
2.00 _ ane process			the start code detector
stop_after_picture	1	0	If the mask register is set to 1 then an interrupt
			can be generated and the start code detector
	~		will stop. See A.11.5.1
i·			stop_after_picture does not reset to 0 after
			the end of a picture has been detected so
			should be cleared directly.

Table A.11.1 Start code detector registers (Sheet 3 of 5)



Register name		Size/Dir.	Roset State	Description
non_aligned_start_event		1	0	When ignore_non_aligned is set to '_start
		₩	<u> </u>	codes that are not byte aligned are ignored
non_aligned_start_mask		1	0	(Vealed as normal data).
Indiana paga shared	<u> </u>	w		When ignore_non_aligned is set to 0 = 251
ignore_non_aligned	1		0	and MPEG start codes will be detected
		w		regardless of byte alignment and the non-
				aligned start event will be generated.
				If the mask register is set to 1 then the event
				will cause an interrupt and the start code
			į	detector will stop. See A.11 6
				If the coding standard is configured as UPEG
				Ignore_non_atigned is ignored and the non-
discard_extension_data		_		aligned start event will never be generated.
	1		1	When these registers are set to 1 extension or
diagond				user data that cannot be decoded by the
discard_user_data	1		1	Spatial Decoder is discarded by the start code
	~~			detector. See A.11.3.3
discard_all_data	1		0	When set to 1 all data and Tokens are
	rw.			discarded by the start code detector. This
				continues until a FLUSH Token is supplied or
				the register is set to 0 directly.
				The FLUSH Token that resets this register is
				discarded and not output by the start code
IDSAT (AGUAGA)		-		detector. See A.11.5.1
insert_sequence_start	1	1		See A.11.7
	~			

Table A.11.1 Start code detector registers (Sheet 4 of 5)



Register name	SizwOli	Roset State	Description
start_code_search	3	5	When this register is set to 0 the stain code
	rw.	,	detector operates normally. When set to a
			higher value the start code detector discards
			data until the specified type of start code is
			delected. When the specified start code is
			detected the register is set to 0 and normal
			operation follows. See A.11.3
start_code_detector_coding_standard	2	0	This register configures the coding standard
	r ~		used by the start code detector. The register
			can be loaded directly or by using a
			CODING_STANDARD Taken.
			Whenever the start code detector generales a
			CODING_STANDARD Token (see
			A.11.7.4 it can es its current
			coding standard configuration. This Token will
			then configure the coding standard used by all
			other parts of the decoder chip-set. See A.21.1
•			and A.11.7
bictrie_unimper	4	0	Each time the stan coded detector detects a
	~		picture start code in the data stream (or the
			H.261 or UPEG equivalent) a
			PICTURE_START Token is generated
			which carries the current value of
			picture_number. This register then
			increments.

Table A.11.1 Start code detector registers (Sheet 5 of 5)

and the second of the second o

Register name	Stro/Dir	Resol State	Description
length_count	16	0	This register contains the current value of the
	rO		JPEG length count. This register is modified
			under the control of the coded data clock and
			should only be read via the MPI when the start
			code detector is stopped.

Table A.11.2 Start code detector test registers

A.11.3 Conversion of start codes to Tokens

In normal operation the function of the Start Code Detector is to identify start codes in the data stream and to then convert them to the appropriate start code Token. In the simplest case, data is supplied to the Start code Detector in a single long DATA Token. The output of the Start Code Detector is a number of shorter DATA Tokens interleaved with start code Tokens.

Alternatively, in accordance with the present invention, 10 the input data to the Start Code Detector could be divided up into a number of shorter DATA Tokens. There is no restriction on how the coded data is divided into DATA Tokens other than that each DATA Token must contain 8 \times n 15 bits where n is an integer.

Other Tokens can be supplied directly to the input of the Start Code Detector. In this case, the Tokens are passed through the Start Code Detector with no processing

to other stages of the Spatial Decoder. These Tokens can only be inserted just before the location of a start code in the coded data.

A.11.3.1 Start code formats

Three different start code formats are recognized by the Start Code Detector of the present invention. This is configured via the register, start_code_detector_coding_standard.

Coding Standard	Start Code Pattern (hex)	Size of start code value		
MPEG	0x00 0x00 0x01 <value></value>	8 bit		
JPEG	0xFF <value></value>	8 bit		
H.261	- 0x00 0x01 <value></value>	4 bit		

Table A.11.3 Start code formats

10 A.11.3.2 Start code Token equivalents

Having detected a start code, the Start Code Detector studies the value associated with the start code and generates an appropriate Token. In general, the Tokens are named after the relevant MPEG syntax. However, one of ordinary skill in the art will appreciate that the Tokens can follow additional naming formats. The coding standard currently selected configures the relationship between start code value and the Token generated. This relationship is shown in Table A.11.4.

	Start Code Value						
Start code Toxen generated	MPEG	H 251	JPEG	UPEG			
	(hex)	(hex)	(hex)	(name)			
PICTURE_START	0x00	0×00	0xOA	; scs			
SLICE_START 4	0x01 to	0x01 to	0xD0 to	AST ₃ to			
	OXAF	OxCC	0xD7	2ST-			
SEQUENCE_START	0x83		0xC8	SOI			
SEQUENCE_END	0x87	İ	0xC9	ECI			
GROUP_START	0x58		0xC0	SCF ₂ ?			
USER_DATA	0x82		0xE0 to	APPo to			
			0xEF	4225			
			0xFE	COM			
EXTENSION_DATA	0x85		0xC8	; JPG			
			0xF0 to	್ರೆ∂G₂ :၁			
			0xFO	JPG _D			
			0xC2 to	AES (
			CxSF				
			0xC1 to	SOF: to			
			0xCB	SCF11			
			0xCC	DAC			
DHT_MARKER			0xC4	онт			
DNL_MARKER			0xDC	ONL			
DQT_MARKER			0x0B	DOT :			
DRI_MARKER			0x00	ORI :			

Table A.11.4 Tokens from start code values

- This Token contains an 8 bit data field which is loaded with a value determined by the start code value.
- b. Indicates start of baseline DCT encoded data.

30

A.11.3_3 Extended features of the coding standards

The coding standards provide a number of mechanisms to allow data to be embedded in the data stream whose use is not currently defined by the coding standard. This might be application specific "user data" that provides extra facilities for a particular manufacturer. Alternatively, it might be "extension data". The coding standards authorities reserved the right to use the extension data to add features to the coding standard in the future.

Two distinct mechanisms are employed. JPEG precedes blocks of user and extension data with marker codes. However, H.261 inserts "extra information" indicated by an extra information bit in the coded data. MPEG can use both these techniques.

In accordance with the present invention, MPEG/JPEG blocks of user and extension data preceded by start/marker codes can be detected by the Start Code Detector.

H.261/MPEG "extra information" is detected by the Huffman decoder of the present invention. See A.14.7, "Receiving Extra Information".

The registers, discard_extension_data and discard_user_data, allow the Start Code Detector to be configured to discard user data and extension data. If this data is not discarded at the Start Code Detector it can be accessed when it reaches the Video Demux see A.14.6, "Receiving User and Extension data".

The Spatial Decoder of the present invention supports the baseline features of JPEG. The non-baseline features of JPEG are viewed as extension data by the Spatial Decoder. So, all JPEG marker codes that precede data for non-baseline JPEG are treated as extension data.

A.11.3.4 JPEG Table definitions

JPEG supports down loaded Huffman and quantizer tables. In JPEG data, the definition of these tables is preceded by the marker codes DNL and DQT. The Start Code Detector generates the Tokens DHT_MARKER and DQT_MARKER when these marker codes are detected. These Tokens indicate to the Video Demux that the DATA Token which follows contains coded data describing Huffman or quantizer table (using the formats described in JPEG).

10 A.11.4 Error detection

5

20

25

30

The Start Code Detector can detect certain errors in the coded data and provides some facilities to allow the decoder to recover after an error is detected (see A.11.8, "Start code searching").

15 A.11.4.1 Illegal JPEG length count

Most JPEG marker codes have a 16 bit length count field associated with them. This field indicates how much data is associated with this marker code. Length counts of 0 and 1 are illegal. An illegal length should only occur following a data error. In the present invention, this will generate an interrupt if illegal_length_count_mask is set to 1.

Recovery from errors in JPEG data is likely to require additional application specific data due to the difficulty of searching for start codes in JPEG data (see A.11.8.1).

A.11.4.2 Overlapping start/marker codes

In the present invention, overlapping start codes should only occur following a data error. An MPEG, byte aligned, overlapping start code is illustrated in Figure 64. Here, the Start Code Detector first sees a pattern that looks like a picture start code. Next the Start Code Detector sees that this picture start code is overlapped with a group start. Accordingly, the Start Code Detector

1. 11

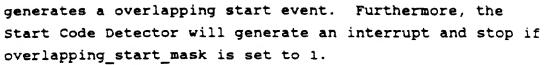
1 19 1

10

15

20

30



It is impossible to tell which of the two start codes is the correct one and which was caused by a data error. However, the Start Code Detector in accordance with the present invention, discards the first start code and will proceed decoding the second start code "as if it is correct" after the overlapping start-code event has been serviced. If there are a series of overlapped start codes, the Start Code Detector will discard all but the last (generating an event for each overlapping start code).

Similar errors are possible in non byte-aligned systems (H.261 or possibly MPEG). In this case, the state of ignore_non_aligned must also be considered. Figure 65 illustrates an example where the first start code found is byte aligned, but it overlaps a non-aligned start code. If ignore_non_aligned is set to 1, then the second overlapping start code will be treated as data by the Start Code Detector and, therefore no overlapping start code event will occur. This conceals a possible data communications error. If ignore_non_aligned is set to 0, however the Start Code Detector will see the second, non aligned, start code and will see that it overlaps the first start code.

25 A.11.4.3 Unrecognized start codes

The Start Code Detector can generate an interrupt when an unrecognized start code is detected (if unrecognized_start_mask = 1). The value of the start code that caused this interrupt can be read from the register start value.

The start code value 0xB4 (sequence error) is used in MPEG decoder systems to indicate a channel or media error. For example, this start code may be inserted into the data by an ECC circuit if it detects an error that it was unable

20

30

to correct.

A.11.4.4 Sequence of event generation

In the present invention, certain coded data patterns (probably indicating an error condition) will cause more than one of the above error conditions to occur within a short space of time. Consequently, the sequence in which the Start Code Detector examines the coded data for error conditions is:

- 1) Non-aligned start codes
- 10 2) Overlapping start codes
 - 3) Unrecognized start codes

Thus, if a non-aligned start code overlaps another, later, start code, the first event generated will be associated with the non-aligned start code. After this event has been serviced, the Start Code Detector's operation will proceed, detecting the overlapped start code a short time later.

The Start Code Detector only attempts to recognize the start code after all tests for non-aligned and overlapping start codes are complete.

A.11.5 Decoder start-up and shutdown

The Start Code Detector provides facilities to allow the current decoding task to be completed cleanly and for a new task to be started.

There are limitations on using these techniques with JPEG coded video as data segments can contain values that emulate marker codes (see A.11.8.1).

A.11.5.1 Clean end to decoding

The Start Code Detector can be configured to generate an interrupt and stop once the data for the current picture is complete. This is done by setting stop_after_picture = 1 and stop_after_picture_mask = 1.

Once the end of a picture passes through the Start Code Detector, a FLUSH Token is generated (A.11.7.2),

=

=

and the second second

15

20

25

30

an intergupt is generated, and the Start Code Detector stops. Note that the picture just completed will be decoded in the normal way. In some applications, however, it may be appropriate to detect the FLUSH arriving at the output of the decoder chip-set as this will indicate the end of the current video sequence. For example, the display could freeze on the last picture output.

When the Start Code Detector stops, there may be data from the "old" video sequence "trapped" in user implemented buffers between the media and the decode chips. the register, discard_all_data, will cause the Spatial Decoder to consume and discard this data. continue until a FLUSH Token reaches the Start Code Detector or discard all data is reset via the microprocessor interface.

Having discarded any data from the "old" sequence the decoder is now ready to start work on a new sequence.

A.11.5.2 When to start discard all mode

The discard all mode will start immediately after a 1 is written into the discard all data register. The result will be unpredictable if this is done when the Start Code Detector is actively processing data.

Discard all mode can be safely initiated after any of the Start Code Detector events (non-aligned start event etc.) has generated an interrupt.

A.11.5.3 Starting a new sequence

If it is not known where the start of a new coded video sequence is within some coded data, then the start code search mechanism can be used. This discards any unwanted data that precedes the start of the sequence. See A.11.8.

A.11.5.4 Jumping between sequences

This section illustrates an application of some of the techniques described above. The objective is to "jump"

20

from one part of one coded video sequence to another. In this example, the filing system only allows access to "blocks" of data. This block structure might be derived from the sector size of a disc or a block error correction system. So, the position of entry and exit points in the coded video data may not be related to the filing system block structure.

The stop_after_picture and discard_all_data mechanisms allow unwanted data from the old video sequence to be discarded. Inserting a FLUSH Token after the end of the last filing system data block resets the discard_all_data mode. The start code search mode can then be used to discard any data in the next data block that precedes a suitable entry point.

15 A.11.6 Byte alignment

As is well known in the art, the different coding schemes have quite different views about byte alignment of start/marker codes in the data stream.

For example, H.261 views communications as being bit serial. Thus, there is no concept of byte alignment of start codes. By setting ignore_non_aligned = 0 the Start Code Detector is able to detect start codes with any bit alignment. By setting non-aligned_start_mask = 0, the start code non-alignment interrupt is suppressed.

In contrast, however, JPEG was designed for a computer environment where byte alignment is guaranteed. Therefore marker codes should only be detected when byte aligned. When the coding standard is configured as JPEG, the register ignore_non_aligned is ignored and the non-aligned start event will never be generated. However, setting ignore_non_aligned = 1 and non_aligned_start_mask = 0 is recommended to ensure compatibility with future products.

MPEG, on the other hand, was designed to meet the needs of both communications (bit serial) and computer (byte

15

20

oriented) systems. Start codes in MPEG data should normally be byte aligned. However, the standard is designed to be allow bit serial searching for start codes (no MPEG bit pattern, with any bit alignment, will look like a start code, unless it is a start code). So, an MPEG decoder can be designed that will tolerate loss of byte alignment in serial data communications.

If a non-aligned start code is found, it will normally indicate that a communication error has previously occurred. If the error is a "bit-slip" in a bit-serial communications system, then data containing this error will have already been passed to the decoder. This error is likely to cause other errors within the decoder. However, new data arriving at the Start Code Detector can continue to be decoded after this loss of byte alignment.

By setting ignore_non_aligned = 0 and non_aligned_start_mask = 1, an interrupt can be generated if a non-aligned start code is detected. The response will depend upon the application. All subsequent start codes will be non-aligned (until byte alignment is restored). Accordingly, setting non_aligned_start_mask = 0 after byte alignment has been lost may be appropriate.

	MPEG	JPEG	H.251	1
ignore_non_aligned	0	1	0	=
non_aligned_start_mask	1	0	0	-

Table A.11.5 Configuring for byte alignment

15

25

30

A.11.7 Automatic Token generation

In the present invention, most of the Tokens output by the Start Code Detector directly reflect syntactic elements of the various picture and video coding standards. addition to these "natural" Tokens, some useful "invented" Tokens are generated. Examples of these proprietary tokens are PICTURE END and CODING STANDARD. Tokens are also introduced to remove some of the syntactic differences between the coding standards and to "tidy up" under error conditions.

This automatic Token generation is done after the serial analysis of the coded data (see Figure 61, "The Start Code Detector"). Therefore the system responds equally to Tokens that have been supplied directly to the input of the Spatial Decoder via the Start Code Detector and to Tokens that have been generated by the Start Code Detector following the detection of start codes in the coded data.

A.11.7.1 Indicating the end of a picture

In general, the coding standards don't explicitly signal the end of a picture. However, the Start Code Detector of the present invention generates a PICTURE END Token when it detects information that indicates that the current picture has been completed.

The Tokens that cause PICTURE END to be generated are: SEQUENCE START, GROUP START, PICTURE START, SEQUENCE END and FLUSH.

A.11.7.2 Stop after picture end option

If the register stop_after_picture is set, then the Start Code Detector will stop after a PICTURE END Token has passed through. However, a FLUSH Token is inserted after the PICTURE END to "push" the tail end of the coded data through the decoder and to reset the system. See A.11.5.1.

20

30

A.11. 7.3 Introducing sequence start for H.261

H.261 does not have a syntactic element equivalent to sequence start (see Table A.11.4). If the register insert_sequence_start is set, then the Start Code Detector will ensure that there is one SEQUENCE_START Token before the next PICTURE_START, i.e., if the Start Code Detector does not see a SEQUENCE_START before a PICTURE_START, one will be introduced. No SEQUENCE_START will be introduced if one is already present.

This function should not be used with MPEG or JPEG.

A.11.7.4 Setting coding standard for each sequence

All SEQUENCE_START Tokens leaving the Start Code
Detector are always preceded by a CODING_STANDARD Token.
This Token is loaded with the Start Code Detector's current coding standard. This sets the coding standard for the entire decoder chip set for each new video sequence.

A.11.8 Start code searching

The Start Code Detector in accordance with the invention, can be used to search through a coded data stream for a specified type of start code. This allows the decoder to re-commence decoding from a specified level within the syntax of some coded data (after discarding any data that precedes it). Applications for this include:

rstart-up of a decoder after jumping into a coded data file at an unknown position (e.g., random accessing). To seek to a known point in the data to assist recovery after a data error.

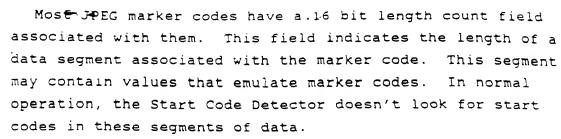
For example, Table A.11.6 shows the MPEG start codes searched, for different configurations of start_code_search. The equivalent H.261 and JPEG start/marker codes can be seen in Table A.11.4.

the state of the s

start_code_search	Start codes searched for	
0 •	Normal operation	
1	: Reserved (will behave as discard data)	
2		
3	sequence start	
start_code_search	Start codes searched for .	
4	group or sequence start	
4 5 ⁵	group or sequence start picture, group or sequence start	

Table A.11.6 Start code search modes

- a. A FLUSH Token places the Start Code Detector in this search mode.
- b. This is the default mode after reset.
- When a non-zero value is written into the start_code_search register, the Start Code Detector will start to discard all incoming data until the specified start code is detected. The start_code_search register will then reset to 0 and normal operation will continue.
- The start code search will start immediately after a non-zero value is written into the start_code_search register. The result will be unpredictable if this is done when the Start Code Detector is actively processing data. So, before initiating a start code search, the Start Code
- Detector should be stopped so no data is being processed. The Start Code Detector is always in this condition if any of the Start Code Detector events (non-aligned start event etc.) has just generated an interrupt.
 - A.11.8.1 Limitations on using start code search with JPEG



If a random access into some JPEG coded data "lands" in such a segment, the start code search mechanism cannot be used reliably. In general, JPEG coded video will require additional external information to identify entry points for random access.

15

20

25

SECTION A.12 Decoder start-up control

A.12.1 Overview of decoder start-up

In a decoder, video display will normally be delayed a short time after coded data is first available. During this delay, coded data accumulates in the buffers in the decoder. This pre-filling of the buffers ensures that the buffers never empty during decoding and, this, therefore ensures that the decoder is able to decode new pictures at regular intervals.

Generally, two facilities are required to correctly start-up a decoder. First, there must be a mechanism to measure how much data has been provided to the decoder. Second, there must be a mechanism to prevent the display of a new video stream. The Spatial Decoder of the invention provides a bit counter near its input to measure how much data has arrived and an output gate near its output to prevent the start of new video stream being output.

There are three levels of complexity for the control of these facilities:

- Output gate always open
 - ·Basic control
 - · Advanced control

With the output gate always open, picture output will start as soon as possible after coded data starts to arrive at the decoder. This is appropriate for still picture decoding or where display is being delayed by some other mechanism.

The difference between basic and advanced control relates to how many short video streams can be accommodated in the decoder's buffers at any time. Basic control is sufficient for most applications. However, advanced control allows user software to help the decoder manage the start-up of several very short video streams.

15

20

25

A.12.2 MPEG video buffer verifier

MPEG describes a "video buffer verifier" (VBV) for constant data rate systems. Using the VBV information allows the decoder to pre-fill its buffers before it starts to display pictures. Again, this pre-filling ensures that the decoder's buffers never empty during decoding.

In summary, each MPEG picture carries a vbv_delay parameter. This parameter specifies how long the coded data buffer of an "ideal decoder" should fill with coded data before the first picture is decoded. Having observed the start-up delay for the first picture, the requirements of all subsequent pictures will be met automatically.

MPEG, therefore, specifies the start-up requirements as a delay. However, in a constant bit rate system this delay can readily be converted to a bit count. This is the basis on which the start-up control of the Spatial Decoder of the present invention operates.

A.12.3 Definition of a stream

In this application, the term stream is used to avoid confusion with the MPEG term sequence. Stream therefore means a quantity of video data that is "interesting" to an application. Hence, a stream could be many MPEG sequences or it could be a single picture.

The decoder start-up facilities described in this chapter relate to meeting the VBV requirements of the first picture in a stream. The requirements of subsequent pictures in that stream are met automatically.

A.12.4 Start-up control registers

Register name	Sitte/Dir	Rusot Stato	Description
startup_access	1	0	Writing 1 to this register requests that the bit
CED_BS_ACCESS	~		counter and gate opening logic stop to allow
			access to their configuration registers
bit_count	8	٥	This bit counter is incremented as coded data
CED_BS_COUNT	~		leaves the start code detector. The number of
bit_count_prescale	3	0	bits required to increment bit_count thice is
CED_BS_PRESCALE	~		approx. 2(bit_count_prescale=') x 512.
			The bit counter starts counting bits after a
			FLUSH Token passes through the oil counter
			It is reset to zero and then stops incrementing
			after the bit count larget has been met.
bit_count_target	8	x	This register specifies the bit count target. A
CED_8S_TARGET	~		target met event is generated whenever the
			following condition becomes true:
			bit_count >= bit_count_target
target_met_event	1	0	When the bit count target is met this event will
SS_TARGET_MET_EVENT	rw		be generated, if the mask register is set to 1
target_met_mask	1	0	then an interrupt can be generated. However
	rw		the bit counter will NOT stop processing data
			This event will occur when the bit counter
			increments to its target, it will also occur if a
			target value is written which is less than or
			equal to the current value of the bit counter
			Writing 0 to bit_count_target will a have
			generate a target met event.

Table A.12.1 Decoder start-up registers

the second secon

Register name	Slze/Dk.	Reset State	Description
counter_flushed_event	1	0	When a FLUSH Token passes through the oil
BS_FLUSH_EVENT	rw		count circuit this event will occur if the mask
counter_flushed_mask	1	0	register is set to 1 then an interrupt can be
	rw		generated and the bit counter will stop
counter_flushed_too_early_ event	1	0	If a FLUSH Token passes through the bit
BS_FLUSH_BEFORE_TARGET_MET_EVENT	₽₩		count cubuit and the bit count target has not
counter_flushed_too_early_mask	1	0	been met this event will occur. If the mask
	rw		register is set to 1 then an interrupt can be
			generated and the bit counter will stop.
			See A.12.10
offchip_queue	1	0	Setting this register to 1 configures the gate
CED_BS_CUEUE	~		opening logic to require microprocessor
			support. When this register is set to 0 the output
			gate control logic will automatically control the
			operation of the output gate.
			See sections A.12.5 and A.12.7.
enable_stream	1	0	When an off-chip queue is in use writing to
CED_BS_ENABLE_NXT_STM	~		enable_stream controls the behaviour of the
			output gate after the end of a stream passes
			through it.
			A one in this register enables the output gate to
			open.
			The register will be reset when an
			accept_enable interrupt is generaled

Table A.12.1 Decoder start-up registers (contd)

10 1 2 2 1 1 1

Register name	Size/Dir.	Resel State	Description	_
accept_enable_event	1	0	This event indicates that a FLUSH Token has	=
SS_STREAM_END_EVENT	~		passed through the output gate (causing it to	:
accept_enable_mask	1	0	close) and that an enable was available to allow	•
	rw		the gate to open.	:
			If the mask register is set to 1 then an interrupt	
			can be generated and the register	
			enable_stream will be reset. See A.12.7.1	;
				_

Table A.12.1 Decoder start-up registers (contd)

The second secon

15

20

25

30

A.12. Dutput gate always open

The output gate can be configured to remain open. This configuration is appropriate where still pictures are being decoded, or when some other mechanism is available to manage the start-up of the video decoder.

The following configurations are required after reset (having gained access to the start-up control logic by writing 1 to startup access):

set offchip queue = 1

set enable stream = 1

registers are set to 0 disabling their interrupts (this is the default state after reset).

(See A.12.7.1 for an explanation of why this holds the output gate open.)

A.12.6 Basic operation

In the present invention, basic control of the start-up logic is sufficient for the majority of MPEG video applications. In this mode, the bit counter communicates directly with the output gate. The output gate will close automatically as the end of a video stream passes through it as indicated by a FLUSH Token. The gate will remain closed until an enable is provided by the bit counter circuitry when a stream has attained its start-up bit count.

The following configurations are required after reset (having gained access to the start-up control logic by writing 1 to startup access):

set bit_count_prescale approximately for the expected range of coded data rates

set counter_flushed_too_early_mask = 1 to enable this
error condition to be detected

Two interrupt service routines are required:

·Video Demux service to obtain the value of

e in the second of the second

10

wbw delay for the first picture in each new stream

· Counter flushed too early service to react to this condition

The video demux (also known as the video parser) can generate an interrupt when it decodes the vbv_delay for a new video stream (i.e., the first picture to arrive at the video demux after a FLUSH). The interrupt service routine should compute an appropriate value for bit_count_target and write it. When the bit counter reaches this target, it will insert an enable into a short queue between the bit counter and the output gate. When the output gate opens it removes an enable from this queue.

· presente

10

30

A.12.6.1 Starting a new stream shortly after another finishes

As an example, the MPEG stream which is about to finish is called A and the MPEG stream about to start is called B. A FLUSH Token should be inserted after the end of A. This pushes the last of its coded data through the decoder and alerts the various sections of the decoder to expect a new stream.

Normally, the bit counter will have reset to zero, A having already met its start-up conditions. After the FLUSH, the bit counter will start counting the bits in stream B. When the Video Demux has decoded the vbv_delay from the first picture in stream B, an interrupt will be generated allowing the bit counter to be configured.

As the FLUSH marking the end of stream A passes through the output gate, the gate will close. The gate will remain closed until B meets its start-up conditions. Depending on a number of factors such as: the start-up delay for stream B and the depth of the buffers, it is possible that B will have already met its start-up conditions when the output gate closes. In this case, there will be an enable waiting in the queue and the output gate will immediately open. Otherwise, stream B will have to wait until it meets its start-up requirements.

25 A.12.6.2 A succession of short streams

The capacity of the queue located between the bit counter and the output gate is sufficient to allow 3 separate video streams to have met their start-up conditions and to be waiting for a previous stream to finish being decoded. In the present invention, this situation will only occur if very short streams are being decoded or if the off-chip buffers are very large as compared to the picture format being decoded).

In Figure 69 stream A is being decoded and the

15

output gate is open). Streams B and C have met their start-up conditions and are entirely contained within the buffers managed by the Spatial Decoder. Stream D is still arriving at the input of the Spatial Decoder.

Enables for streams B and C are in the queue. So, when stream A is completed B will be able to start immediately. Similarly C can follow immediately behind B.

If A is still passing through the output gate when D meets its start-up target an enable will be added to the queue, filling the queue. If no enables have been removed from the queue by the time the end of D passes the bit counter (i.e., A is still passing through the output gate) no new stream will be able to start through the bit counter. Therefore, coded data will be held up at the input until A completes and an enable is removed from the queue as the output gate is opened to allow B to pass through.

A.12.7 Advanced operation

In accordance with the present invention, advanced control of the start-up logic allows user software to infinitely extend the length of the enable queue described in A.12.6, "Basic operation". This level of control will only be required where the video decoder must accommodate a series of short video streams longer than that described in A.12.6.2, "A succession of short streams".

In addition to the configuration required for Basic operation of the system, the following configurations are required after reset (having gained access to the start-up control logic by writing 1 to start_up access):

set offchip_queue = 1

set accept_enable_mask = 1 to enable interrupts

when an enable has been removed from the queue

set target_met_mask = 1 to enable interrupts

when a stream's bit count target is met

Two-additional interrupt service routines are required:

- accept enable interrupt
- · Target met interrupt
- 5 When a target met interrupt occurs, the service routine should add an enable to its off-chip enable queue.

A.12.7.1 Output gate logic behavior

Writing a 1 to the enable_stream register loads an enable into a short queue.

When a FLUSH (marking the end of a stream) passes through the output gate the gate will close. If there is an enable available at the end of the queue, the gate will open and generate an accept_enable_event. If accept_enable_mask is set to one, an interrupt can be generated and an enable is removed from the end of the queue (the register enable stream is reset).

However, if accept_enable_mask is set to zero, no interrupt is generated following the accept_enable_event and the enable is NOT removed from the end of the queue.

This mechanism can be used to keep the output gate open as described in A.12.5.

A.12.8 Bit counting

The bit counter starts counting after a FLUSH Token passes through it. This FLUSH Token indicates the end of the current video stream. In this regard, the bit counter continues counting until it meets the bit count target set in the bit_count_target register. A target met event is then generated and the bit counter resets to zero and waits for the next FLUSH Token.

The bit counter will also stop incrementing when it reaches it maximum count (255).

A.12.9 Bit count prescale

In the present invention, $2^{\frac{(h_1, count_present-1)}{2}} \times 512$ bits are

10

15

required to increment the bit counter once. Furthermore, bit_count_prescale is a 3 bit register than can hold a value between 0 and 7.

n	Range (bits)	Resolution (bits)				
0	0 to 252144	1024				
1	0 to 524288	2048				
7	0 to 31457280	122880				

Table A.12.2 Example bit counter ranges

The bit count is approximate, as some elements of the video stream will already have been Tokenized (e.g., the start codes) and, therefore includes non-data Tokens.

A.12.10 Counter flushed too early

If a FLUSH token arrives at the bit counter before the bit count target is attained, an event is generated which can cause an interrupt (if counter_flushed_too_early_mask = 1). If the interrupt is generated, then the bit counter circuit will stop, preventing further data input. It is the responsibility of the user's software to decide when to open the output gate after this event has occurred. The output gate can be made to open by writing 0 as the bit count target. These circumstances should only arise when trying to decode video streams that last only a few pictures.

30

SECTION A.13 Buffer Management

The Spatial Decoder manages two logical data buffers: the coded data buffer (CDB) and the Token buffer (TB).

The CDB buffers coded data between the Start Code
Detector and the input of the Huffman decoder. This
provides buffering for low data rate coded video data. The
TB buffers data between the output of the Huffman decoder
and the input of the spatial video decoding circuits
(inverse modeler, quantizer and DCT). This second logical
buffer allows processing time to include a spread so as to
accommodate processing pictures having varying amounts of
data.

Both buffers are physically held in a single off-chip DRAM array. The addresses for these buffers are generated by the buffer manager.

A.13.1 Buffer manager registers

The Spatial Decoder buffer manager is intended to be configured once immediately after the device is reset. In normal operation, there is no requirement to reconfigure the buffer manager.

After reset is removed from the Spatial Decoder, the buffer manager is halted (with its access register, buffer_manager_access, set to 1) awaiting configuration. After the registers have been configured,

25 buffer_manager_access can be set to 0 and decoding can commence.

Most of the registers used in the buffer manager cannot be accessed reliably while the buffer manager is operating. Before any of the buffer manager registers are accessed buffer_manager_access must be set to 1. This makes it essential to observe the protocol of waiting until the value 1 can be read from buffer_manager_access. The time taken to obtain and release access should be taken into

consideration when polling such registers as cdb_full and cdb_empty to monitor buffer conditions.

Register name	SkøDir	Rosol Stato	Description
buffer_manager_access	1	1	This access bit stops the operation of the burder manager so that is
	~		various registers can be accessed reliably. See A 6.4.1
			Note: this access register is unusual as its default state after reset is
		· 	1. I.e. after reset the buffer manager is halted awaiting configuration
			via the microprocessor interface.
Register name	Slza/Dir.	Rosot State	Description
buffer_manager_keyhole_address	6	x	Keyhole access to the extended address space used for the outler
	r w		manager registers shown below. See A 5 4 3 for more
buffer_manager_keyhole_data	8	x	information about accessing registers through a keynole
	~		
buffer_limit	18	x	This specifies the overall size of the DF + L after interned to the
	۳٦		Spatial Decoder, All buffer addresses are in place and MCD mis buffer
			size and so will wrap round within the \mathbb{D}^{2n+1} provided
Cdb_base	18	x	These registers point to the base of the first data (ccb) and Token
tb_base	r ₩		(tb) buffers.
cdb_length	18	x	These register is specify the length (i.e. size) of the coded data (cdb)
tb_length	۳۰		and Toke (tb) buffers.
cdb_read	18	x	These registers hold an offset from the buffer hase and indicate
tb_read	ro		where data il be read from next.
cdb_number	18	x	These resisters show how much data is a entity held in the puriers
tb_number	ro		
cab_fuli	1	x	There ingisters will be set to illif the line ad data (cdd) or Token ib
tb_full	ro		מעליי ואי פער מילים מעליים איז איז איז איז איז איז איז איז איז איז
cdb_empty	1	x	The egisters will be set to 1 if the Loded data (cdb) or Token 12
	1		

Table A.13.1 Buffer manager registers (contd)

15

20

25

A.13. L.1 Buffer manager pointer values

Typically, data is transferred between the Spatial Decoder and the off_chip DRAM in 64 byte bursts (using the DRAM's fast page mode). All the buffer pointers and length registers refer to these 64 byte (512 bit) blocks of data. So, the buffer manager's 18 bit registers describe a 256 k block linear address space (i.e., 128 Mb).

The 64 byte transfer is independent of the width (8, 16 or 32 bits) of the DRAM interface.

10 A.13.2 Use of the buffer manager registers

The Spatial Decoder buffer manager has two sets of registers that define two similar buffers. The buffer limit register (buffer_limit) defines the physical upper limit of the memory space. All addresses are calculated modulo this number.

Within the limits of the available memory, the extent of each buffer is defined by two registers: the buffer base (cdb_base and tb_base) and the buffer length (cdb_length and tb_length). All the registers described thus far must be configured before the buffers can be used.

The current status of each buffer is visible in 4 registers. The buffer read register (cdb_read and tb_read) indicates an offset from the buffer base from which data will be read next. The buffer number registers (cdb_number and tb_number) indicate the amount of data currently held by buffers. The status bits cdb_full, tb_full, cdb_empty and tb_empty indicate if the buffers are full or empty.

As stated in A.13.1.1, the unit for all the above mentioned registers is a 512 bit block of data.

Accordingly, the value read from cdb_number should be multiplied by 512 to obtain the number of bits in the coded data buffer.

A.13.3 Zero buffers

Still picture applications (e.g., using JPEG) that do

HP.

15

20

not have a "real-time" requirement will not need the large off-chip buffers supported by the buffer manager. In this case, the DRAM interface can be configured (by writing 1 to the zero_buffers register) to ignore the buffer manager to provide a 128 bit stream on-chip FIFO for the coded data buffer and the Token buffers.

The zero buffers option may also be appropriate for applications which operate working at low data rates and with small picture formats.

Note: the zero_buffers register is part of the DRAM interface and, therefore, should be set only during the post-reset configuration of the DRAM interface.

A.13.4 Buffer operation

The data transfer through the buffers is controlled by a handshake Protocol. Hence, it is guaranteed that no data errors will occur if the buffer fills or empties. If a buffer is filled, then the circuits trying to send data to the buffer will be halted until there is space in the buffer. If a buffer continues to be full, more processing stages "up steam" of the buffer will halt until the Spatial Decoder is unable to accept data on its input port. Similarly, if a buffer empties, then the circuits trying to remove data from the buffer will halt until data is available.

As described in A.13.2, the position and size of the coded data and Token buffer are specified by the buffer base and length registers. The user is responsible for configuring these registers and for ensuring that there is no conflict in memory usage between the two buffers.

production of the second of the second

SECTION A.14 Video Demux

The Video Demux or Video parser as it is also called, completes the task of converting coded data into Tokens started by the Start Code Detector. There are four main processing blocks in the Video Demux: Parser State Machine, Huffman decoder (including an ITOD), Macroblock counter and ALU.

The Parser or state machine follows the syntax of the coded video data and instructs the other units. The Huffman decoder converts variable length coded (VLC) data into integers. The Macroblock counter keeps track of which section of a picture is being decoded. The ALU performs the necessary arithmetic calculations.

A.14.1 Video Demux registers

Register name	Sizo/Dlr	Rosel State	Description .
demux_access	1	0	This access bit stops the operation of the Video Demux so that its
CED_H_CTAL(T)	~		vanous registers can be accessed reliably. See A 6 4 1
huffman_error_code	3		When the Video Demux stops following the generation of a
CED_H_CTRL(5.4)	ιο		huffman_event interrupt request this 3 bit register holds a value indicating
			why the interrupt was generated. See A.14 5.1
parser_error_code	8		When the Video Demux stops following the generation of a parser_event
CED_H_DMUX_EAR	ro		interrupt request this 8 bit register holds a value indicating why the
			interrupt was generated. See A.14 5.2
demux_keyhole_address	12	x	Keyhole access to the Video Demux's extended address scace. See
CED_H_KEYHCLE_ADDR	~		A.6.4.3 for more information about accessing registers
demux_keynole_data	8	x	through a keyhole.
CED_H_KEYHOLE	۲ ۰		Tables A.14.2, A.14.3 and A.14.4 describe the registers that can be
			accessed via the keyhole.

Table A.14.1 Top level Video Demux registers

15

10

Regist ername	Siza/Dir	Resot State	Description
dummy_last_picture	1	0	When this register is set to 1 the Video Demux will generate information
CED_H_ALU_REGO	r*		for a "dummy" intra picture as the last picture of an MPEG sequence
r_ram_cantrol			This function is useful when the Temporal Decoder is configured for
r_dummy_last_frame_bit			automatic picture re-ordering (see A 18.3.5, "Picture sequence re-
,244			ordering", to flush the last P or I picture out of the Temporal
			Decoder
			No "dummy" picture is required if
			- the Temporal Decoder is not configured for re-ordering
			another MPEG sequence will be decoded immediately (as this will also
			flush out the last picture)
			• the coding standard is not MPEG
field_info	1	0	When this register is set to 1 the first byte of any MPEG
CED_H_ALU_REG0	~		extra_information_picture is placed in the FIELD_INFO Token. See
r_rom_control			A.1471 -
r_fleid_info_bit			
continue	1	0	This register allows user software to control how much extra, user or
CED_H_ALU_REGO	~		extension data it wants to receive when is it is detected by the decoder.
r_ram_control			See A.14.6 and A.14.7
r_continue_bit			
rom_revision	8		Immediately following reset this holds a copy of the microcope ACM
CED_H_ALU_REGI	ro		revision number.
r_rom_revision			This register is also used to present to control software data values read
			from the coded data. See A.14 6, "Receiving User and Extension data"
			and A.14.7, "Receiving Extra Information",

Table A.14.1 Top level Video Demux registers (contd)

11 9

Register name	Siza/Dir	Reset State	Description
huffman_event	1	0	A Huffman event is generated if an error is found in the coded data. See
	~		A.14.5.1 for a description of these events,
huffman_mask	1 rw	0	If the mask register is set to 1 then an interrupt can be generated and the Video Demux will stop. If the mask register is set to 0 then no interrupt is generated and the Video Demux will attempt to recover from the error
parser_event	1	0	A Parser event can be in responce to errors in the coded data or to the
	~		arrival of information at the Video Demux that requires software
parser_mask	1	0	intervention. See A.14.5.2 for a description of these events
	rw		If the mask register is set to 1 then an interrupt can be generated and the Video Demux will stop. If the mask register is set to 0 then no interrupt is
			generated and the Video Demux will attempt to continue.

Table A.14.1 Top level Video Demux registers (contd)

Register name	Slze/Dlr.	Reset State	Description
component_name_0	8	x	During JPEG operation the register component_name_n holds an 3 bit value
component_name_1	rw		indicating (to an application) which colour component has the component (0 a
component_name_2			
component_name_3			
horiz_pels	16	×	These registers hold the horizontal and vertical dimensions of the video being
-	rw		decoded in pixels.
vert_pels	16	x	See section A.14.2
horiz_macroplocks	16	x	These registers hold the horizontal and vertical dimensions of the viceo being
	~		decoded in macroblocks.
vert_macroblocks	16	x	See section A.14.2
	~		2-0 32500 A 1-2

Table A.14.2 video demux picture construction registers

Rosot Stato Stru/Dk. Register name Description max_h These registers hold the macrobiock width and height in blocks (8 x 8 pixers) 2 The values 0 to 3 indicate a width/height of 1 to 4 blocks max_v 2 X See section A.14.2 max_component_id The values 0 to 3 indicate that 1 to 4 different viceo components are currently being decoded. rw See section A.14.2 8 During JPEG operation this register holds the parameter NI (number of image ı rw components in frame). blocks_h_0 2 For each of the 4 colour components the registers blocks_h_n and X $blocks_h_1$ blocks_v_n hold the number of blocks horizontally and vertically in a blocks_h_2 macroblock for the colour component with component ID in blocks_h_3 See section A.14.2 blocks_v_0 2 x blocks_v_1 æ blocks_v_2 blocks_v_3 tq_0 2 x The two bit value held by the register tq_n describes which inverse 10_1 Quantisation table is to be used when decoding data with component (D) a rw. tq_2 :0_3

Table A.14.2 Video demux picture construction registers (contd)

15

A.14.1-1- Register loading and Token generation

Many of the registers in the Video Demux hold values that relate directly to parameters normally communicated in the coded picture/video data. For example, the horiz_pels register corresponds to the MPEG sequence header information, horizontal_size, and the JPEG frame header parameter, X. These registers are loaded by the Video Demux when the appropriate coded data is decoded. These registers are also associated with a Token. For example, the register, horiz_pels, is associated with Token, HORIZONTAL_SIZE. The Token is generated by the Video Demux when (or soon after) the coded data is decoded. The Token can also be supplied directly to the input of the Spatial Decoder. In this case, the value carried by the Token will configure the Video Demux register associated with it.

			295
Regist ane	Stze/Olt.	Rosel State	Description
ac_huff_0	2		The two bit value held by the register do_huff_n dascribes which Huffman
dc_huff_1	~		decoding table is to be used when decoding the DC doefficients of cata with
dc_huff_2			componentID n.
de_huff_3			Similarly ac_huff_ri describes the table to be used when decoding AC
ac_huff_0	2		coefficients.
ac_huff_1	rw		Baseline JPEG requires up to two Huffman tables per scan. The only factes
ac_huff_2			implemented are 0 and 1
ac_huff_3			implemented are of and i
dc_bits_0[15:0]	8		Each of these is a table of 16, eight bit values. They provide the BiTS
dc_bits_1[15:0]	rw		information (see JPEG Huffman table specification) which form part of the
ac_bits_0[15:0]	8		description of two DC and two AC Huffman tables.
ac_bits_1[15:0]	~~		See section A.14 3.1
dc_huffvai_0(11:0)	8	1	Each of these is a table of 12, eight bit values. They provide the HUFFIAL
dc_huffval_1[11:0]	₩		information (see JPEG Huffman table specification) which form part of the
			description of two DC Huffman tables
			See section A.14 3.1
ac_huffval_0[161:0]	8		Each of these is a table of 162, eight bit values. They provide the HUFFIAL
ac_huffval_1[161:0]	rw		information (see JPEG Huffman table specification) which form part of the
			description of two AC Huffman tables.
			See section A.14.3.1
dc_25555_0	6		These 8 bit registers hold values that are "special cased" to accelerate the
dc_zssss_1	rw		decoding of certain frequently used JPEG YLCs.
ac_eob_0	8		dc_ssss - magnitude of DC coefficient is 0
ac_eob_1	~		ac_eob - end of block
ac_zri_0	8	-	ac_zrl - run of 16 zeros
ac_zrl_1	rw		

Table A.14.3 Video demux Huffman table registers

11 (0.2)



Register name	Size/Dir	Reset State	D esc ription .
buffer_size	10		This register is loaded when decoding MPEG data with a value indicating the
	~		size of VBV buffer required in an ideal decoder.
			This value is not used by the decoder chips. However, the value it noics may
			be useful to user software when configuring the coded data buffer size and to
			determine whether the decoder is capable of decoding a particular MPEB cala
			file.
pel_aspect	4		This register is loaded when decoding MPES data with a value incidering the
	r*		pel aspect ratio. The value is a 4 bit integer that is used as an index into a
			lable defined by MPEG.
			See the MPEG standard for a definition of this table
			This value is not used by the decoder chips. However, the value if hords may
			be useful to user software when configuring a display or output device
bit_rate	18		This register is loaded when decoding MPEG data with a value incideing the
	~		coded data rate.
			See the MPEG standard for a definition of this value.
			This value is not used by the decoder chips. However, the value it holds may
			be useful to user software when configuring the decoder stari-up registers.
pic_rate	4		This register is loaded when decoding MPEG data with a value indicating the
	~		picture rate.
			See the MPEG standard for a definition of this value.
			This value is not used by the decoder chips. However, the value if ho as max
1			be useful to user software when configuring a display or output device
constrained	1		This register is loaded when decoding MPEG data to indicate if the coded data
	~		meets MPEG's constrained parameters.
			See the MPEG standard for a definition of this flag.
1			This value is not used by the decoder chips. However, the value 1 colds flat
	ļ		be useful to user software to determine whether the decoder is capable of
			decoding a particular MPEG data file.

Table A.14.4 Other Video Demux registers

H1 11

Register name	Stra/Dk.	Resat State	Description			
picture_type	2		During MPEG operation this register holds the picture type of the picture being			
	rw		decoded.			
h_251_pic_type	8		This register is loaded when decoding H.251 data, it holds information about			
	₩		the picture format.			
			7 6 5 4 3 2 1 0 r r s d r q r r			
			Flags:			
			s - Split Screen Indicator			
			d - Document Camera			
			1 - Freeze Picture Release			
			This value is not used by the decoder chips. However, the information should			
			be used when configuring horiz_pels, vert_pels and the display or output			
			device.			
broken_closed	2	1	During MPEG operation this register holds the broken_link and closed_goo			
•	rw		information for the group of pictures being decoded.			
			7 6 5 4 3 2 1 0 r r r r r r c b Flags:			
			c - closed_gop			

Table A.14.4 Other Video Demux registers (contd)



Register name	Slze/Dk.	Nesel State	Description				
prediction_mode	5		During MPEG and H.251operation this register holds the current value of				
	~	İ	prediction mode.				
			7 6 5 4 3 2 1 0 r r r h y x b r Flags. h - enable H.261 loop filter y - reset backward vector prediction				
vbv_delay	16		This register is loaded when decoding MPEG data with a value indicating the				
	r ~		minimum start-up delay before decoding should start.				
		!	See the MPEG standard for a definition of this value.				
			This value is not used by the decoder chips. However, the value it holds may				
			be useful to user software when configuring the decoder start-up registers				
pic_number	a		This register holds the picture number for the pictures that is currently being				
	~		decoded by the Video Demux. This number was generated by the start code				
			detector when this picture arrived there.				
			See Table A.11.2 for a description of the picture number				
dummy_last_picture	1	0	These registers are also visible at the top level. See Table A.14.1				
	rw						
field_info	1	0					
	rw	_					
continue	1	0					
100 1000		-	1				
rom_revision	8						
coding_standard	2	+-	This register is loaded by the CODING_STANDARD Token to configure				
	ro		the Video Demux's mode of operation.				
			See section A.21.1				

Table A.14.4 Other Video Demux registers (contd)

Register name	Slzø/Dlr.	Resel State	Description
restart_interval	8		This register is loaded when decoding JPEG data with a value indicating the
	~		minimum start-up delay before decooing should start.
		- Annual Control	See the MPEG standard for a definition of this value.

Table A.14.4 Other Video Demux registers (contd)

register	Token	standard	comment
component_name_n	COMPONENT_NAME	PONENT_NAME JPEG in co	
		MPEG	not used in standard
		H.251	
horiz_peis	HORIZONTAL_SIZE	MPEG	in coded data.
ven_pels	VERTICAL_SIZE	JPEG	
		H 251	automatically derived from picture
			туре
horiz_macroblocks	HORIZONTAL_MBS	MPEG	control software must derive from
vert_macroblocks	VERTICAL_MBS	JPEG	horizontal and vertical picture size
		H.251	automatically derived from picture
			type.
max_h	DEFINE_MAX_SAMPLING	MPEG	control software must configure.
max_v			Sampling structure is fixed by
			standard
		JPEG	in coded data.
		H.251	automatically configured for 4.2.3
			video

Table A.14.5 Register to Token cross reference



register	Token	standard	comment
max_component_ld	MAX_COMP_ID	MPEG	control software must configure
			Sampling structure is fixed by
			standard
		JPEG	in coded data.
		H.261	automatically configured for 4.2.0
			video
tq_0	JPEG_TABLE_SELECT	JPEG	in coded data.
tq_1		MPEG	not used in standard
tq_2		H.261	
tq_3			
blocks_h_0	DEFINE_SAMPLING	MPEG	control software must configure.
blacks_h_1			Sampling structure is fixed by
blocks_h_2			standard.
blocks_h_3		JPEG	in coded data.
		H.261	automatically configured for 4.2.0
blocks_v_0			video
blocks_v_1			
blocks_v_2			
blocks_v_3			
dc_huff_0	in scan header data	JPEG	in coded data.
dc_huff_1	MPEG_DCH_TABLE	MPEG	control software must configure.
dc_huff_2		H.261	not used in standard.
dc_huff_3			
ac_huff_0	in scan header data	JPEG	in coded data.
ac_huff_1		MPEG	not used in standard.
ac_huff_2		H.261	
ac_huff_3			

Table A.14.5 Register to Token cross reference (contd)



register	Token	standard	comment
ic_bits_0(15.0)	in DATAToxen following	JPEG	in coded data.
ic_bits_1[15.0]	DHT_MARKER Token		
(c_nuffvai_0(11.0)		MPEG	control software must configure
-		H.251	not used in standard
sc_huffval_1(11:0)			1
ic_zssss_0			
c_zssss_1			
ic_bits_0[15.0]	in DATAToken following	JPEG	in coded data
ac_bits_1(15:0)	DHT_MARKER Token		
and the second second second		MPEG	not used in standard
ac_huffval_0(161:0)		H.251	
ac_huffval_1[161:0]			
ac_epb_0			
ac_eob_1			
0			
ac_zrl_0			
ac_zri_1	LVOV BUTETO CITE		
buffer_size	VBV_BUFFER_SIZE	MPEG	in coded data.
		JPEG	not used in standard
	DEL ASDECT	H.251	
pel_aspect	PEL_ASPECT	MPEG	in coded data.
		JPEG H.251	not used in standard
hit rate	BIT_RATE	MPEG	in coded data.
bit_rate	SII_IINIE	JPEG	not used in standard
		H.261	
pic_rate	PICTURE_RATE	MPEG	in coded cata.
F1-0-21-00-10		JPEG	not used in standard
		H.251	-
constrained	CONSTRAINED	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
picture_type	PICTURE_TYPE	MPEG	in coded data.
/		JPEG	not used in standard
		H.261	⊣

Table A.14.5 Register to Token cross reference (contd)

register Token standard comment BROKEN_CLOSED MPEG broken_closed in coded data. JPEG not used in standard H251 PREDICTION_MODE prediction_mode MPEG in coded data. JPEG not used in standard H251 PICTURE_TYPE h_261_pic_type MPEG not relevant JPEG (when standard is H.261) H.251 in coded data. VBV_DELAY MPEG vbv_delay in coded data. **JPEG** not used in standard H.251 Carried by: MPEG Generated by start code detector pic_number JPEG PICTURE_START H.251 CODING_STANDARD MPEG coding_standard configured in start code by control **JPEG** software detector. H251

Table A.14.5 Register to Token cross reference (contd)

A.14.2 Picture structure

In the present invention, picture dimensions are
described to the Spatial Decoder in 2 different units:
pixels and macroblocks. JPEG and MPEG both communicate
picture dimensions in pixels. Communicating the dimensions
in pixels determine the area of the buffer that contains
the valid data; this may be smaller than the total buffer
size. Communicating dimensions in macroblocks determines
the size of buffer required by the decoder. The macroblock
dimensions must be derived by the user from the pixel

dimensions. The Spatial Decoder registers associated with this information are: horiz_pels, vert_pels, horiz_macroblocks and vert macroblocks.

The Spatial Decoder registers, blocks_h_n, blocks_v_n, max_h, max_v and max_component_id specify the composition of the macroblocks (minimum coding units in JPEG). Each is a 2 bit register than can hold values in the range 0 to 3. All except max_component_id specify a block count of 1 to 4. For example, if register max_h holds 1, then a macroblock is two blocks wide. Similarly, max_component_id specifies the number of different color components involved.

	2:1:1	4:2:2	4:2:0	1:1:1
max_h	1	1	1	0
max_v	0	1	t	0
max_component_id	2	2	2	2
blocks_h_0	1	1	1	0
blocks_h_1	0	0	0	0
blocks_h_2	0	0	0	0
blocks_h_3	x	x	x	x
blocks_v_0	0	t	1	0 1
biocks_v_1	0	1	0	0
blocks_v_2	0	1	O	0
blocks_v_3	x	x	x	x

Table A.14.6 Configuration for various macroblock formats

A.14. > Huffman tables

A.14.3.1 JPEG style Huffman table descriptions

In the invention, Huffman table descriptions are provided to the Spatial decoder via the format used by JPEG to communicate table descriptions between encoders and decoders. There are two elements to each table description: BITS and HUFFVAL. For a full description of how tables are encoded, the user is directed to the JPEG specification.

10 A.14.3.1.1 BITS

5

20

25

30

BITS is a table of values that describes how many different symbols are encoded with each length of VLC. Each entry is an 8 bit value. JPEG permits VLCs with up to 16 bits long, so there are 16 entries in each table.

The BITS[0] describes how many different 1 bit VLCs exist while BITS[1] describes how many different 2 bit VLCs exist and so forth.

A.14.3.1.2 HUFFVAL

HUFFVAL is table of 8 bit data values arranged in order of increasing VLC length. The size of this table will depend on the number of different symbols that can be encoded by the VLC.

The JPEG specification describes in further detail how Huffman coding tables can be encoded or decoded into this format.

A.14.3.1.3 Configuration by Tokens

In a JPEG bitstream, the DHT marker precedes the description of the Huffman tables used to code AC and DC coefficients. When the Start Code Detector recognizes a DHT marker, it generates a DHT_MARKER Token and places the Huffman table description in the following DATA Token (see A.11.3.4).

Configuration of AC and DC coefficient Huffman tables within the Spatial Decoder can be achieved by supplying

DATA And DHT_MARKER Tokens to the input of the Spatial Decoder while the Spatial Decoder is configured for JPEG operation. This mechanism can be used for configuring the DC coefficient Huffman tables required for MPEG operation, however, the coding standard of the Spatial Decoder must be set to JPEG while the tables are down loaded.

E 7 5 5 4 3 2 1 1 0	Token Name	
1 0 0 0 1 0 1 1 0 1 1	CODING_STANDARD	
0 0 0 0 0 0 0 0 0 1	1 = JPEG	
0 0 0 0 1 1 1 1 0 0	DHT_MARKER	
1 0 0 0 0 0 1 x x	DATA	
1	Th - Value indicating which Huffman table is to be loaded, JPEG allows 4	
	tables to be downloaded.	
	Values 0x00 and 0x01 specify DC coefficient coding tables 0 and 1	
	Values 0x10 and 0x11 specifies AC coefficient coding tables 0 and 1	k e
	니 - 16 words carrying BITS information	3 2
		o alto
		ned t
	V _{ij} - Words carrying HUFFVAL information (the	robos
:	number of words depends on the number of different	so po
ennnnnn	symbols).	ince ca
	e - the extension bit will be 0 if this is the endof the DATA Token or 1 if	This sequence can be repeated to allow bryonal tables to be described in a shiple Tokum
	another table description is contained in the same DATA Token.	1 1

Table A.14.7 Huffman table configuration via Tokens

A.14.3-14 Configuration by MPI

The AC and DC coefficient Huffman tables can also be written directly to registers via the MPI. See Table A.14.3.

- The registers dc_bits_0[15:0] and dc_bits_1[15:0] hold the BITS values for tables 0x00 and 0x01.

 The registers ac_bits_0[15:0] and ac_bits_1[15:0] hold the BITS values for tables 0x10 and 0x11.

 The registers dc_huffval_0[11:0] and
- dc_huffval_1[11:0] hold the HUFFVAL values for tables 0x00 and 0x01.
 - The registers ac_huffval_0[161:0] and ac_huffval_1[161:0] hold the HUFFVAL values for tables 0x10 and 0x11.
- 15 A.14.4 Configuring for different standards

The Video Demux supports the requirements of MPEG, JPEG and H.261. The coding standard is configured automatically by the CODING_STANDARD Token generated by the Start Code Detector.

20 A.14.4.1 H.261 Huffman tables

All the Huffman tables required to decode H.261 are held in ROMs within the Spatial Decoder and more particular in the parser state machine of the Video demux and, therefore require no user intervention.

25 A.14.4.2 H.261 Picture structure

H.261 is defined as supporting only two picture formats: CIF and QCIF. The picture format in use is signalled in the PTYPE section of the bitstream. When this data is decoded by the Spatial Decoder, it is placed in the

h_261_pic_type registers and the PICTURE_TYPE Token. In addition, all the picture and macroblock construction registers are configured automatically.

The information in the various registers is also placed into their related Tokens (see Table A.14.5),

and this ensures that other decoder chips (such as the Temporal Decoder) are correctly configured.

A.14.4.3 MPEG Huffman tables

The majority of the Huffman coding tables required to decode MPEG are held in ROMs within the Spatial Decoder (again, in the parser state machine) and, thus, require no user intervention. The exceptions are the tables required for decoding the DC coefficients of Intral macroblocks. Two tables are required, one for chroma the other for luma.

These must be configured by user software before decoding begins.

macroblock construction	CIF/	picture construction	CIF	QCIF
max_h	1	horiz_pels	352	176
max_v	1	vert_pels	288	144
max_component_id	2	horiz_macroblocks	22	11
blocks_h_0	1	vert_macroblocks	18	9
blocks_h_1	0			
blocks_h_2	0			i
blocks_v_0	1			
blacks_v_1	0			
blocks_v_2	0			

Table A.14.8 Automatic settings for H.261

Table A.14.10 shows the sequence of Tokens required to configure the DC coefficient Huffman tables within the Spatial Decoder. Alternatively, the same results can be obtained by writing this information to registers via the MPI.

The registers dc_huff_n control which DC coefficient Huffman tables are used with each color component. Table

A.14.9 shows how they should be configured for MPEG operation. This can be done directly via the MPI or by using the MPEG_DCH_TABLE Token.

dc_huff_0	0
dc_huff_1	1
dc_huff_2	1
dc_huff_3	x

Table A.14.9 MPEG DC Huffman table selection via MPI

E	[0:7]	Token Name
1	0x15	CODING_STANDARD
0	0x01	1 = JPEG
0	0x1C	DHT_MARKER
1	0x04	DATA (could be any colour component, 0 is used in this example)
1	0x00	0 indicates that this Huffman table is DC coefficient coding table 0

Table A.14.10 MPEG DC Huffman table configuration

5

10 - 21

E	[0.7]	Token Name
1	0×00	16 words carrying BITS information describing a total of 9
1	0×02	different VLCs:
1	0×03	
1	0×01	2, 2 bit codes
1	0x01	3. 3 bit codes
1	0x01	1, 4 bit codes
1	0x01	1, 5 bit codes
1	0×00	1, 6 bit codes
1	0x00	1,7 bit codes
1	0×00	
1	0×00	If configuring via the MPI rather than with Tokens these values would be
1	0×00	written into the do_bits_0(15:0) registers.
1	0x00	
1	0x00	
1	0×00	- -
1	0x00	
1	0x01	9 words carrying HUFFVAL information
1	0x02	If configuring via the MPI rather than with Tokens these values would be
1	0x00	
1	0×03	written into the dc_huffval_0(11:0) registers.
1	0x04	
1	0x05	
1	0x06	
1	0x07	
0	80x0	

Table A.14.10 MPEG DC Huffman table configuration (contd)

Ε	[7.0]	Token Name
0	0x1C	DHT_MARKER
1	0x04	DATA (could be any colour component, 0 is used in this example)
1	0x01	fundicates that this Huffman table is DC coefficient coding table f
1	0x00	16 words carrying BITS information describing a total of 9
1	0x03	different VLCs:
:	0x01	3. 2 bit codes
1	0x01	
: 11	0x01	1, 3 bit codes
1	0x01	1, 4 bit codes
1	0x01	1, 5 bit codes
1	0x01	1, 6 bit codes
1	0×00	1, 7 bit codes
1	0×00	1, 8 bit codes
	0x00	
1	0x00	If configuring via the MPI rather than with Tokens these values would be
1	0x00	written into the dc_bits_1[15:0] registers.
1	0x00	
1	0×00	
	0x00	LOursele acquire HIJESVA1 information
	0x00	9 words carrying HUFFVAL information
1	0x01	If configuring via the MPI rather than with Tokens these values would be
1	0x02	written into the dc_huffvai_1[11:0] registers.
1	0x04	
1	0x05	-
1	0x06	-
1	0x07	-
0	0x08	-
1	0x04	MPEG_DCH_TABLE
0	0×∞	Configure so table 0 is used for component 0
1	0x05	MPEG_DCH_TABLE
0	0×01	Configure so table 1 is used for component 1
1	0x06	
0	0x01	Configure so table 1 is used for component 2
<u> </u>	-!!	

Table A.14.10 MPEG DC Huffman table configuration (contd)

15

Ε	[7.0]	Token Name
1	0x15	CODING_STANDARD
٥	0x02	2 = JPEG

Table A.14.10 MPEG DC Huffman table configuration (contd)

A.14.4.4 MPEG Picture structure

The macroblock construction defined for MPEG is the same as that used by H.261. The picture dimensions are encoded in the coded data.

For standard 4:2:0 operation, the macroblock characteristics should be configured as indicated in Table A.14.8. This can be done either by writing to the registers as indicated or by applying the equivalent Tokens (see Table A.14.5) to the input of the Spatial Decoder.

The approach taken to configure picture dimensions will depend upon the application. If the picture format is known before decoding starts, then the picture construction registers listed in Table A.14.8 can be initialized with appropriate values. Alternatively, the picture dimensions can be decoded from the coded data and used to configure the Spatial Decoder. In this case the user must service the parser error ERR_MPEG_SEQUENCE, see A.14.8, "Changes at the MPEG sequence layer".

20

20

A.14.4:5- JPEG

Within baseline JPEG, there are a number of encoder options that significantly alter the complexity of the control software required to operate the decoder. In general, the Spatial Decoder has been designed so that the required support is minimal where the following condition is met:

Number of color components per frame is less than $5(N, \le 4)$

10 A.14.4.6 JPEG Huffman tables

Furthermore, JPEG allows Huffman coding tables to be down loaded to the decoder. These tables are used when decoding the VLCs describing the coefficients. Two tables are permitted per scan for decoding DC coefficients and two for the AC coefficients.

There are three different types of JPEG file:
Interchange format, an abbreviated format for compressed image data, and an abbreviated format for table data. In an interchange format file there is both compressed image data and a definition of all the tables (Huffman, Quantization etc.) required to decode the image data. The abbreviated image data format file omits the table definitions. The abbreviated table format file only contains the table definitions.

25 The Spatial Decoder will accept all three formats.

However, abbreviated image data files can only be decoded if all the required tables have been defined. This definition can be done via either of the other two JPEG file types, or alternatively, the tables could be set-up by user software.

If each scan uses a different set of Huffman tables, then the table definitions are placed (by the encoder) in the coded data before each scan. These are automatically loaded by the Spatial Decoder for use during this and any

٠

15

30

subsequent scans.

To improve the performance of the Huffman decoding, certain commonly used symbols are specially cased. These are: DC coefficient with magnitude 0, end of block AC coefficients and run of 16 zero AC coefficients. The values for these special cases should be written into the appropriate registers.

A.14.4.6.1 Table selection

The registers dc_huff_n and ac_huff_n control which AC and DC coefficient Huffman tables are used with which color component. During JPEG operation, these relationships are defined by the TD_j and Ta_j fields of the scan header syntax.

A.14.4.7 JPEG Picture structure

There are two distinct levels of baseline JPEG decoding supported by the Spatial Decoder: up to 4 components per frame $(N_f{\le}4)$ and greater than 4 components per frame $(N_f{>}4)$. If $N_f{>}4$ is used, the control software required becomes more complex.

A.14.4.7.1 Nf<4

The frame component specification parameters contained in the JPEG frame header configure the macroblock construction registers (see Table A.14.8) when they are decoded. No user intervention is required, as all the specifications required to decode the 4 different color components as defined.

For further details of the options provided by JPEG the reader should study the JPEG specification. Also, there is a short description of JPEG picture formats in § A.16.1.

A.14.4.7.2 JPEG with more than 4 components

The Spatial Decoder can decode JPEG files containing up to 256 different color components (the maximum permitted by JPEG). However, additional user intervention is required if more than 4 color component are to be decoded. JPEG only allows a maximum of 4 components in any scan.

only allows a maximum of 4 components in any scan.

A.14.4.8 Non-standard variants

As stated above, the Spatial Decoder supports some picture formats beyond those defined by JPEG and MPEG.

JPEG limits minimum coding units so that they contain no more than 10 blocks per scan. This limit does not apply to the Spatial Decoder since it can process any minimum coding unit that can be described by blocks_h_n, blocks_v_n, max_h and max_v.

MPEG is only defined for 4:2:0 macroblocks (see Table A.14.8). However, the Spatial Decoder can process three other component macroblock structures, (e.g., 4:2:2.

A.14.5 Video events and errors

The Video Demux can generate two types of events: parser events and Huffman events. See A.6.3, "Interrupts", for a description of how to handle events and interrupts.

A.14.5.1 Huffman events

Huffman events are generated by the Huffman decoder.

The event which is indicated in huffman_event and huffman_mask determines whether an interrupt is generated. If huffman_mask is set to 1, an interrupt will be generated and the Huffman decoder will halt. The register huffman_error_code[2:0] will hold a value indicating the cause of the event.

If 1 is written to huffman_event after servicing the interrupt, the Huffman decoder will attempt to recover from the error. Also, if huffman_mask was set to 0 (masking the interrupt and not halting the Huffman decoder) the Huffman decoder will attempt to recover from the error

30 automatically.

A.14.5.2 Parser events

Parser events are generated by the Parser. The event is

15

indicated in parser_event. Thereafter, parser_mask determines whether an interrupt is generated. If parser_mask is set to 1, an interrupt will be generated and the Parser will halt. The register parser_error_code[7:0] will hold a value indicating the cause of event.

If 1 is written to huffman_event after servicing the interrupt, the Huffman decoder will attempt to recover from the error. Also, if huffman_mask was set to 0 (masking the interrupt and not halting the Huffman decoder) the Huffman decoder will attempt to recover form the error automatically.

If 1 is written to parser_event after servicing the interrupt, the Parser will start operation again. If the event indicated a bitstream error, the Video Demux will attempt to recover from the error.

If parser_mask was set to 0, the Parser will set its event bit, but will not generate an interrupt or halt. It will continue operation and attempt to recover from the error automatically.

hu	ffman_error_c	ode	5
[2]	গে	[0]	Description
0	0	0	No error. This error should not occur during
			normal operation
×	0	1	Failed to find terminal code in VLC within 15
·			bits.
X	1	0	Found serial data when Token expected
×	1	1	Found Token when serial data expected
1	X	×	information describing more than 54
			coefficients for a single block was decoded
			indicating a bitstream error. The block output by
			the Video Demux will contain only 64
			coefficients.

Table A.14.11 Huffman error codes

parser_error_code(7:0)	Description
0x00	ERR_NO_ERROR
	No Parser error has occured, this event should not occur during normal operation
0x10	ERR_EXTENSION_TOKEN
	An EXTENSION_DATA Token has been detected by the Parser. The detection of
	this Token should preceed a DATA Token that contains the extension data. See A 14.5
0×11	ERR_EXTENSION_DATA
	Following the detection of an EXTENSION_DATA Token, a DATA Token
	containing the extension data has been detedcted. See A.14.6
0x12	ERR_USER_TOKEN
	A USER_DATA Token has been detected by the Parser. The detection of this Token
	should preceed a DATA Token that contains the user data. See A,14,5
0x13	ERR_USER_DATA
	Following the detection of a USER_DATA Token, a DATA Token containing the user
	data has been detedcted. See A.14 6
0x20	ERR_PSPARE
	H.261 PSARE information has been detected see A.14,7

Table A.14.12 Parser error codes (Sheet 1 of 5)

parser_error_cade(Z:0)	
	Description
0x21	ERR_GSPARE
	H.261 GSARE information has been detected see A 14 7
Cx22	ERR_PTYPE
	The value of the H.251 picture type has changed. The register h_261_pic_type can be
	inspected to see what the new value is.
0x30	ERR_JPEG_FRAME
0.24	
0x31	ERR_JPEG_FRAME_LAST
0x32	ERR_JPEG_SCAN
	Picture size or Ns changed
0x33	ERR_JPEG_SCAN_COMP
	Component Change !
0x34	ERR_DNL_MARKER
0x40	EAR_MPEG_SEQUENCE
	·
	One of the parameters communicated in the MPEG sequence layer has changed. See
	A.14.8
0x41	ERR_EXTRA_PICTURE
	MPEG extra_information_picture has been detected see A.14.7
0x42	ERR_EXTRA_SLICE
	MPEG extra_information_slice has been detected see A 14.7
) 0x43	ERR_VBV_DELAY
)	The VBV_DELAY parameter for the first picture in a new MPEG video sequence has
	been detected by the Video Demux. The new value of delay is available in the register
	vbv_delay.
	The first picture of a new sequence is defined as the first picture after a sequence end
	FLUSH or reset.
0x80	ERR_SHORT_TOKEN
	An incorrectly formed Token has been detected. This error should not occur during
	ahie A 14 12 Parser error codes (Short 2 of 5)

Table A.14.12 Parser error codes (Sheet 2 of 5)

parser_effor_code(7:0)	Description
0x90	ERR_H261_PIC_END_UNEXPECTED
	During H.251 operation the end of a picture has been encountered at an unexpected
	position. This is likely to indicate an error in the coded data.
Cx91	ERR_GN_BACKUP
	During H.261 operation a group of blocks has been encountered with a group number
	less than that expected. This is likely to indicate an error in the coded data
Cx92	ERR_GN_SKIP_GOB
	During H.251 operation a group of blocks has been encountered with a group number
	greater than that expected. This is likely to indicate an error in the coded data
0xA0	ERR_NBASE_TAB
	During JPEG operation there has been an attempt to down load a Huffman lable that is
	not supported by baseline JPEG (baseline JPEG only supports tables 0 and 1 for
	entropy coding).
OxA1	ERR_QUANT_PRECISION
	During JPEG operation there has been an attempt to down load a quantisation table that
	is not supported by baseline JPEG (baseline JPEG only supports 8 bit precision in
	quantisation (ables).
0xA2	ERR_SAMPLE_PRECISION
	During JPEG operation there has been an attempt to specify a sample precision greater
	than that supported by baseline JPEG (baseline JPEG only supports 8 bit precision).
0xA3	ERR_NBASE_SCAN
	One or more of the JPEG scan header parameters Ss. Se. Ah and Al is set to a value no:
	supported by baseline JPEG (indicating spectral selection and/or successive
	approximation which are not supported in baseline JPEG).
0xA4	ERR_UNEXPECTED_DNL
	During JPEG operation a DNL marker has been encountered in a scan that is not the
	first scan in a frame.
0xA5	ERA_EOS_UNEXPECTED
	During JPEG operation an EOS marker has been encountered in an unexpected place

Table A.14.12 Parser error codes (Sheet 3 of 5)

rit min of

parser_egor_cpde[7:0]	Description
0xA6	ERR_RESTART_SKIP
	During JPEG operation a restart marker has been encountered either in in an
	unexpected place or the value of the restart marker is unexpected. If a restart marker is
	not found when one is expected the Huffman event "Found senal data when Token
	expected" will be generated.
0x80	ERR_SKIP_INTRA
	During MPEG operation, a macro block with a macro block accress increment greater
	than I has been found within an intra (I) picture. This is illegal and probably indicates a
	bilstream error
0x81	ERA_SKIP_DINTRA
	During MPEG operation, a macro block with a macro block address increment greater
	than I has been found within an DC only (D) picture. This is illegal and probably
	indicates a bitstream error.
0x82	ERR_BAO_MARKER
	During MPEG operation, a marker bit did not have the expected value. This is propably
	indicates a bitstream error.
0x83	ERA_D_MBTYPE
	During MPEG operation, within a DC only (D) picture, a macroblock was found with a
	macroblock type other than 1. This is illegal and probably indicates a bitstream error.
0x84	ERR_D_MBEND
	During MPEG operation, within a DC only (D) picture, a macroblock was found with 0 in
	it's end of macroblock bit. This is illegal and probably indicates a bitstream error.
0xB5	ERR_SVP_BACKUP
	During MPEG operation, a slice has been encountered with a slice vertical position less
	than that expected. This is likely to indicate an error in the coded data
0x36	ERR_SVP_SKIP_ROWS
	During MPEG operation, a slice has been encountered with a slice vertical cosition
	greater than that expected. This is likely to indicate an error in the coded ta's
0x97	ERR_FST_MBA_BACKUP
	During MPEG operation, a macroblock has been encountered with a macroblock
	address less than that expected. This is likely to indicate an error in the coded cara

Table A.14.12 Parser error codes (Sheet 4 of 5)

and the second

carsar_arror.	320
0xSb	ERR_FST_MBA_SKIP
	During MPEG operation, a macroblock has been encountered with a macro block
	address greater than that expected. This is likely to indicate an error in the coded data.
0x89	ERR_PICTURE_END_UNEXPECTED
	During MPEG operation, a PICTURE_END Token has been encountered in an
	unexpected place. This is likely to indicate an error in the coded data
0xE0 0xEF	Errors reserved for internal test programs
0xE0	ERR_TST_PROGRAM
	Mysteriously arrived in the test program
CxE1	ERR_NO_PROGRAM
	If the test program is not compiled in
0xE2	ERR_TST_END
	End of Test
0xF0 0xFF	Reserved errors
0×F0	ERR_UCODE_ADDR
	fell off the end of the world
0xF1	ERR_NOT_IMPLEMENTED

Table A.14.12 Parser error codes (Sheet 5 of 5)

Each standard uses a different sub-set of the defined Parser error codes.

Token Name	MPEG	JPEG	H.251
ERR_NO_ERROR	1	1	1
ERR_EXTENSION_TOKEN	1	/	
ERR_EXTENSION_DATA	1	1	
ERR_USER_TOKEN	/	1	<u> </u>
ERR_USER_DATA	1	1) 1
ERR_PSPARE			/
ERR_GSPARE			/
ERR_PTYPE .			/
ERR_JPEG_FRAME		/	1
ERR_JPEG_FRAME_LAST		/	
ERR_JPEG_SCAN		1	<u> </u>

Table A.14.13 Parser error codes and the different standards

5



Token Name	MPEG	JPEG	H.251
ERR_JPEG_SCAN_COMP			
ERR_DNL_MARKER		1	<u> </u>
ERR_MPEG_SEQUENCE	/		1
ERR_EXTRA_PICTURE	1 /		<u> </u>
ERR_EXTRA_SLICE	1		<u> </u>
ERR_VBV_DELAY			<u>!</u>
ERR_SHORT_TOKEN	1		
ERR_H251_PIC_END_UNEXPECTED			/
ERR_GN_BACKUP			/
ERR_GN_SKIP_GOB		1	/
ERR_NBASE_TAB	1	/	
ERR_QUANT_PRECISION	1	1	
ERR_SAMPLE_PRECISION			
ERR_NBASE_SCAN		/	
ERR_UNEXPECTED_DNL		1	
ERR_EOS_UNEXPECTED			
ERR_RESTART_SKIP		/	
ERR_SKIP_INTRA	1		
ERR_SKIP_DINTRA	1		
ERR_BAD_MARKER	1		
ERR_D_MBTYPE	1		<u>_</u>
ERR_D_MBEND	1		
ERR_SVP_BACKUP	1		
ERR_SVP_SKIP_ROWS	1		
ERR_FST_MBA_BACKUP	1	<u> </u>	
ERR_FST_MBA_SKIP	/		
ERR_PICTURE_END_UNEXPECTED	1		
ERR_TST_PROGRAM	1	1	/
ERR_NO_PROGRAM	1	/	/
ERR_TST_END	1	1	/
ERR_UCODE_ADDR	/	/	/
ERR_NOT_IMPLEMENTED	/	/	/

Table A.14.13 Parser error codes and the different standards (contd)

A.14. Exceiving User and Extension data

MPEG and JPEG use similar mechanisms to embed user and extension data. The data is preceded by a start/marker code. The Start Code Detector can be configured to delete this data (see A.11.3.3) if the application has no interest in such data.

A.14.6.1 Identifying the source of the data

The Parser events, ERR_EXTENSION_TOKEN and ERR USER TOKEN, indicate the arrival of the EXTENSION DATA or USER DATA Token at the Video Demux. If these Tokens have been generated by the Start Code Detector, (see A.11.3.3) they will carry the value of the start/marker code that caused the Start Code Detector to generate the Token (see Table A.11.4). This value can be read by reading the rom_revision register while servicing the Parser interrupt. The Video Demux will remain halted until 1 is written to parser event (see A.6.3, "Interrupts"). A.14.6.2 Reading the data

The EXTENSION_DATA and USER_DATA Tokens are expected to be immediately followed by a DATA Token carrying the 20 extension or user data. The arrival of this DATA Token at the Video Demux will generate either an ERR EXTENSION DATA or an ERR_USER_DATA Parser event. The first byte of the DATA Token can be read by reading the rom revision register 25 while servicing the interrupt.

The state of the Video Demux register, continue, determines behavior after the event is cleared. If this register holds the value 0, then any remaining data in the DATA Token will be consumed by the Video Demux and no events will be generated. If the continue is set to 1, an event will be generated as each byte of extension or user data arrives at the Video Demux. This continues until the DATA Token is exhausted or continue is set to 0.

17 LINE AL

NOTE: -

5

25

30

- 1) The first byte of the extension/user data is always presented via the rom_revision register regardless of the state of continue.
- 2) There is no event indicating that the last byte of extension/user data has been read.

A.14.7 Receiving Extra Information

H.261 and MPEG allow information extending the coding standard to be embedded within pictures and groups of blocks (H.261) or slices (MPEG). The mechanism is different from that used for extension and user data (described in Section A.14.6). No start code precedes the data and, thus, it cannot be deleted by the Start Code Detector.

During H.261 operation, the Parser events ERR_PSPARE and ERR_GSPARE indicate the detection of this information. The corresponding events during MPEG operation are

20 ERR_EXTRA_PICTURE and ERR_EXTRA_SLICE.

When the Parser event is generated, the first byte of the extra information is presented through the register, rom revision.

The state of the Video Demux register, continue, determines behavior after the event is cleared. If this register holds the value 0, then any remaining extra information will be consumed by the Video Demux and no events will be generated. If the continue is set to 1, an event will be generated as each byte of extra information arrives at the Video Demux. This continues until the extra information is exhausted or continue is set to 0.

NOTE:

1) The first byte of the extension/user data is always presented via the rom revision

10

15

- register regardless of the state of continue.
- 2) There is no event indicating that the last byte of extension/user data has been read.

A.14.7.1 Generation of the FIELD INFO Token

During MPEG operation, if the register field_info is set to 1, the first byte of any extra_information_picture is placed in the FIELD_INFO Token. This behavior is not covered by the standardization activities of MPEG. Table A.3.2 shows the definition of the FIELD INFO Token.

If field_info is set to 1, no Parser event will be generated for the first byte of extra_information_picture. However, events will be generated for any subsequent bytes of extra_information_picture. If there is only a single byte of extra_information_picture, no Parser event will occur.

A.14.8 Changes at the MPEG sequence layer

The MPEG sequence header describes the following characteristic of the video about to be decoded:

horizontal and vertical size

pixel aspect ratio

· picture rate

coded data rate

25 video buffer verifier buffer size

If any of these parameters change when the Spatial Decoder decodes a sequence header, the Parser event ERR MPEG SEQUENCE will be generated.

A.14.8.1 Change in picture size

If the picture size has changed, the user's software should read the values in horiz_pels and vert_pels and compute new values to be loaded into the registers horiz_macroblocks and vert macroblocks.

15

25

30

SECTION A.15 Spatial Decoding

In accordance with the present invention, the spatial decoding occurs between the output of the Token buffer and the output of the Spatial Decoder.

There are three main units responsible for spatial decoding: the inverse modeler, the inverse quantizer and the inverse discrete cosine transformer. At the input to this section (from the Token buffer) DATA Tokens contain a run and level representation of the quantized coefficients.

At the output (of the inverse DCT) DATA Tokens contain 8x8 blocks of pixel information.

A.15.1 The Inverse Modeler

DATA Tokens in the Token buffer contain information about the values of quantized coefficients and the number of zeros between the coefficients that are represented. The Inverse Modeler expands the information about runs of zeros so that each DATA Token contains 64 values. At this point, the values in the DATA Tokens are quantized coefficients.

The inverse modelling process is the same regardless of the coding standard currently being used. No configuration is required.

For a better understanding of the modelling and inverse modelling function all requirements the reader can examine any of the picture coding standards.

A.15.2 Inverse Quantizer

In an encoder, the quantizer divides down the output of the DCT to reduce the resolution of the DCT coefficients. In a decoder, the function of the inverse quantizer is to multiply up these quantized DCT coefficients to restore them to an approximation of their original values.

A.15.2.1 Overview of the standard quantization schemes

There are significant differences in the quantization

15

20

2.5

schemes used by each of the different coding standards. To obtain a detailed understanding of the quantization schemes used by each of the standards the reader should study the relevant coding standards documents.

The register iq_coding_standard configures the operation of the inverse quantizer to meet the requirements of the different standards. In normal operation, this coding register is automatically loaded by the CODING_STANDARD Token. See section A.21.1 for more information about coding standard configuration.

The main difference between the quantization schemes is the source of the numbers by which the quantized coefficients are multiplied. These are outlined below. There are also detail differences in the arithmetic operations required (rounding etc.), which are not described here.

A.15.2.1.1 H.261 lO overview

In H.261, a single "scale factor" is used to scale the coefficients. The encoder can change this scale factor periodically to regulate the data rate produced. Slightly different rules apply to the "DC" coefficient in intra coded blocks.

A.15.2.1.2 JPEG 10 overview

Baseline JPEG allows for a picture that contains up to 4 different color components in each scan. For each of these 4 color components, a 64 entry quantization table can be specified. Each entry in these tables is used as the "scale" factor for one of the 64 quantized coefficients.

The values for the JPEG quantization tables are contained in the coded JPEG data and will be loaded automatically into the quantization tables.

A.15.2.1.3 MPEG 10 overview

MPEG uses both H.261 and JPEG quantization techniques. Like JPEG, 4 quantization tables, each with 64 entries, can

15

be used. - However, use of the tables is quite different.

Two "types" of data are considered: intra and non-intra. A different table is used for each data type. Two "default" tables are defined by MPEG. One is for use with intra data and the other with non-intra data (see Table A.15.2 and Table A.15.3). These default tables must be written into the quantization table memory of the Spatial Decoder before MPEG decoding is possible.

MPEG also allows two "down loaded" quantization tables. One is for use with intra data and the other with non-intra data. The values for these tables are contained in the MPEG data stream and will be loaded into the quantization table memory automatically.

The value output from the tables is modified by a scale factor.

A.15.2.2 Inverse quantizer registers

Register name	Siza/Dlr.	Reset State	Description
Id_access	1	0	This access bit stops the operation of the inverse quantiser so that its
	~		various registers can be accessed reliably. See A.6.4.1
id_coding_standard	2	0	This register configures the coding standard used by the inverse
	۲۰۰۰		quantiser. The register can be loaded directly or by a
			CODING_STANDARD Token, See A.21 1
iq_keyhole_address	8	x	Keyhole access to the which holds the 4 quantiser tables. See A 5 4 3
	~		for more information about accessing registers through a
iq_keyhole_data	8	×	keyhole.
	~		

Table A.15.1 Inverse quantizer registers

Tell continues on

15

In the present invention, the iq_access register must be set before the quantization table memory can be accessed. The quantization table memory will return the value zero if an attempt is made to read it while iq_access is set to 0.

A.15.2.3 Configuring the inverse quantizer

In normal operation, there is no need to configure the inverse quantizer's coding standard as this will be automatically configured by the CODING STANDARD Token.

For H.261 operation, the quantizer tables are not used. No special configuration is required. For JPEG operation, the tables required by the inverse quantizer should be automatically loaded with information extracted from the coded data.

MPEG operation requires that the default quantization tables are loaded. This should be done while iq_access is set to 1. The values in Table A.15.2 should be written into locations 0x00 to 0x3F of the inverse quantizer's extended address space (accessible through the keyhole registers iq_keyhole_address and iq_keyhole_data).

Similarly, the values in Table A.15.3 should be written into locations 0x40 to 0x7F of the inverse quantizer's extended address space.

- A	W _{i,0} °	i	W, 0	/.	. W.o	,	W.o
0	8	16	27	32	29	48	35
1	16	17	27	33	29	49	38
2	16	18	25	34	27	50	38
3	19	19	26	35	27	51	40
4	16	20	26	36	29	52	
5	19	21	26	37	29	53	40
6	22	22	27	38	32	54	40
7	22	23	27	39	32	55	48 48
8	22	24	27	40	34	56	46
9	22	25	29	41	34	57	46
10	22	26	29	42	37	58	56
11	22	27	29	43	38	59	56
12	26_	28	34	44	37	60	58
13	24	29	34	45	35	61	69
14	26	30	34	46	35	62	69
15	27	31	29	47	34	63	83

Table A.15.2 Default MPEG table for intra coded blocks

- a. Offset from start of quantization table memory
- b. Quantization table value.

-, -	Will	'	144	11	;	1	
	+	"	Wi,1	'	W,	,	w.
0	16	16	16	32	1 16	48	16
1	16	17	16	33	16	49	16
2	16	18	15	34	16	50	15
3	16	19	16	35	16	51	16
4	16	20	16	36	16	52	.5
5	16	21	16	37	16		1
6	16	22	16	38	16	53	15
7	16	23	16	39	16	54	'5
a	16	24	:6	40		55	15
9	16	25	16	41	16	56	16
10				l	16	57	15
	15	25	16	42	15	58	:5
11	16	27	16	43	16	59	1 15
12	16	28	16	44	16	60	15
13	16	29	16	45	16	61	15
14	16	30	16	46	16	62	15
15	16	31	16	47	16	63	15

Table A.15.3 Default MPEG table for non-intra coded blocks

A.15.2.4 configuring tables from Tokens

As an alternative to configuring the inverse quantizer tables via the MPI, they can be initialized by Tokens. These Tokens can be supplied via either the coded data portor the MPI.

The QUANT_TABLE Token is described in Table A.3.2. It has a two bit field tt which specifies which of the 4 (0 to 3) table locations is defined by the Token. For MPEG operation, the default definitions of tables 0 and 1 need to be loaded.

A.15.2.5 quantization table values

For both JPEG and MPEG, the quantization table entries are 8 bit numbers. The values 255 to 1 are legal. The value 0 is illegal.

A.15.2.6 Number ordering of quantization tables

The quantization table values are used in "zig-zag" scan order (see the coding standards). The tables should be viewed as a one dimensional array of 64 values (rather than a 8x8 array). The table entries at lower addresses correspond to the lower frequency DCT coefficients.

When quantization table values are carried by a QUANT_TABLE Token, the first value after the Token header is the table entry for the "DC" coefficient.

A.15. 2.7 Inverse quantizer test registers

Register name	Sua/Dir.	Raset State	Description
iq_quant_scale	5		This register holds the current value of the quantisation scale factor $\langle t \rangle_S$
	rw		loaded by the QUANT_SCALE Token. This is not used during JPES
			operation.
iq_component	2	i	This register holds the two bit component ID taken from the most recent
	~		DATA Token head. This value is involved in the selection of the
			quantiser table.
			The register will also hold the table ID after a QUANT_TABLE Token
			arrives to load the table.
lq_prediction_mode	2		This holds the two LSBs of the most recent PREDICTION_MODE
	~		Token.
iq_peg_indirection	8		This register relates the two bit component ID number of a DATA Token
	~		to the table number of the quantisation table that should be used.
			Bits 1:0 specify the table number that will be sued with component 0
			Bits 3:2 specify the table number that will be sued with component 1
			Bits 5:4 specify the table number that will be sued with component 2
			Bits 7:6 specify the table number that will be sued with component 3
			This register is loaded by JPEG_TABLE_SELECT Tokens.
iq_mpeg_indirection	2	0	This two bit register records whether to use default or down loaced
	~		quantisation tables with the intra and non-intra data.
			A 0 in the bit position indicates that the default table should be used $A(t)$
			indicates that a down loaded table should be used.
			Bit 0 refers to intra data. Bit 1 refers to non-intra data. This register is
			normally loaded by the Token MPEG_TABLE_SELECT

Table A.15.4 Inverse quantiser test registers

A.15. > Inverse Discrete Cosine Transform

The inverse discrete transform processor of the present invention meets the requirements set out in CCITT recommendation H.261, the IEEE specification P1180 and complies with the requirements described in current draft revision of MPEG.

The inverse discrete cosine transform process is the same regardless of which coding standard is used. No, configuration by the user is required.

There are two events associated with the inverse 10 discrete transform processor.

Register name	Size/Dir.	Heset State	Description
ldct_too_lew_event	1	0	The Inverse DCT requires that all DATA Tokens contain exactly 54
	rw		values. If less than 64 values are found then the too-few event wall be
idct_too_few_mask	1	0	generated. If the mask register is set to 1 then an interrupt can be
	~		generated and the inverse DCT will halt.
			This event should only occur following an error in the coded data.
idct_too_many_event	1	0	The Inverse DCT requires that all DATA Tokens contain exactly 64
	۲₩		values. If more than 64 values are found then the too-many event will be
idct_too_many_mask	1	a	generated. If the mask register is set to 1 then an interrupt can be
	rw		generated and the inverse DCT will half.
Company of the second of the s			This event should only occur following an error in the coded data.

Table A.15.5 Inverse DCT event registers

For a better understanding of the DCT and inverse DCT function the reader can examine any of the picture coding 15 standards.

SECTION A.16 Connecting to the output of Spatial Decoder

The output of the Spatial Decoder is a standard Token Port with 9 bit wide data words. See Section A.4 for more information about the electrical behavior of the interface.

The Tokens present at the output will depend on the coding standard employed. By way of example, this section of the disclosure looks at the output of the Spatial Decoder when configured for JPEG operation. This section also describes the Token sequence observed at the output of the Temporal Decoder during JPEG operation as the Temporal Decoder doesn't modify the Token sequence that results from decoding JPEG.

However, MPEG and H.261 both require the use of the Temporal Decoder. See section A.19 for information about connecting to the output of the Temporal Decoder when configured for MPEG and H.261 operation.

Furthermore, this section identifies which of the Tokens are available at the output of the Spatial Decoder and which are most useful when designing circuits to display that output. Other Tokens will be present, but are not needed to display the output and, therefore, are not discussed here.

This section concentrates on showing:

25 How the start and end of sequences can be identified.

·How the start and end of pictures can be identified.

How to identify when to display the picture.

How to identify where in the display the picture data should be placed.

30

5

10

15

20

10

15

20

25

30

A.16.1 Structure of JPEG pictures

This section provides an overview of some features of the JPEG syntax. Please refer to the coding standard for full details.

JPEG provides a variety of mechanisms for encoding individual pictures. JPEG makes no attempt to describe how a collection of pictures could be encoded together to provide a mechanism for encoding video.

The Spatial Decoder, in accordance with the present invention, supports JPEG's baseline sequential mode of operation. There are three main levels in the syntax: Image, Frame and Scan. A sequential image only contains a single frame. A frame can contain between 1 and 256 different image (color) components. These image components can be grouped, in a variety of ways, into scans. Each scan can contain between 1 and 4 image components (see Figure 81 "Overview of JPEG baseline sequential structure").

If a scan contains a single image component, it is non-interleaved, if it contains more than one image component, it is an interleaved scan. A frame can contain a mixture of interleaved and non-interleaved scans. The number of scans that a frame can contain is determined by the 256 limit on the number of image components that a frame can contain.

Within an interleaved scan, data is organized into minimum coding units (MCUs) which are analogous to the macroblock used in MPEG and H.261. These MCUs are raster ordered within a picture. In a non-interleaved scan, the MCU is a single 8x8 block. Again, these are raster organized.

The Spatial Decoder can readily decode JPEG data containing 1 to 4 different color components. Files describing greater numbers of components can also be

decoded. However, some reconfiguration between scans may be required to accommodate the next set of components to be decoded.

A.16.2 Token sequence

The JPEG markers codes are converted to an analogous MPEG named Token by the Start Code Detector (see Table A.11.4, see Fig. 82 "Tokenized JPEG picture").

SECTION A.17 Temporal Decoder.

- · 30 MH, operation
- Provides temporal decoding for MPEG & H.261 video decoders
- .H.261 CIF and QCIF formats
- 5 MPEG video resolutions up to 704x480, 30 Hz, 4:2:0
 - · Flexible chroma sampling formats
 - · Can re-order the MPEG picture sequence
 - · Glue-less DRAM interface
 - ·Single +5V supply
- 10 208 pin PQFP package
 - · Max. power dissipation 2.5W
 - · Uses standard page mode DRAM

The Temporal Decoder is a companion chip to the Spatial Decoder. It provides the temporal decoding required by

15 H.261 and MPEG.

decoded.

20

30

The Temporal Decoder implements all the prediction forming features required by MPEG and H.261. With a single 4 Mb DRAM (e.g., 512 k x 8) the Temporal Decoder can decode CIF and QCIF H.261 video. With 8 Mb of DRAM (e.g., two 256 k x 16) the 704 x 480, 30Hz, 4:2:0 MPEG video can be

The Temporal Decoder is not required for Intra coding schemes (such as JPEG). If included in a multi-standard decoder, the Temporal Decoder will pass decoded JPEG pictures through to its output.

Note: The above values are merely illustrative, by way of example and not necessarily by way of limitation, of one embodiment of the present invention. It will be appreciated that other values and ranges may also be used without departing from the invention.

A.17. Temporal Decoder Signals

Signal Name	1/0	Pin Number	Description			
in_data(8:0]	1	173, 172, 171, 169, 168, 167, 166, 164,	Input Port. This is a standard two wire			
_		163	interface normally connected to the			
n_extn		174	Output Port of the Spatial Decoder			
in_valid	1	162				
in_accept	0	161	See sections A.4 and A.18.1			
enable(1:0)	11	125, 127	Micro Processor Interface MPP			
r w	1	125				
addr(7:0)	1	137, 136, 135, 133, 132, 131, 130, 128				
data[7:0]	0	152, 151, 149, 147, 145, 143, 141, 140	See A.6.1 on page 59.			
िव	0	154				
DRAM_data(31:0)	1/0	15, 17, 19, 20, 22, 25, 27, 30, 31, 33, 35,	DRAM Interface.			
		38, 39, 42, 44, 47, 49, 57, 59, 61, 63, 66,				
		68, 70, 72, 74, 75, 79, 81, 83, 84, 85				
DRAM_addr(10:0)	0	184, 186, 188, 189, 192, 193, 195, 197,	See section A.5.2			
_ , ,		199, 200, 203				
RAS	0	11				
CAS(3:0)	0	2, 4, 6, 8				
WE	0	12				
ŌĒ	0	204				
DRAM_enable	1	112				
out_data(7:0)	0	89, 90, 92, 93, 94, 95, 97, 98	Output Port. This is a standard two wire			
out_extn	0	87	interface.			
out_valid	0	99	See sections 4.4 and A.15			
out_accept	1	100	See sections A 4 and A 13			
tck ·	1	115	JTAG port.			
taı	1	116	See section A.9			
tdo	0	120				
tms	ı	117				
trst	ı	121				
decoder_clock	!	177	The main decoder clock. See			
			Table A.7 2			
reset	1	150	Reset.			

Table A.17.1 Temporal Decoder signals

Signal Name	vo	Pin Num.	Description
tph0ish	1	122	If override = 1 then tpn0ish and tph1ish are incuts for the on-on-o
tph1ish	ı	123	two phase clock.
overnde	,	110	For normal operation set overfide = 0, tph0ish and tph1ish are
			ignored (so connect to GND or V _{DD}).
chiptest	1	111	Set chiptest = 0 for normal operation.
tioop	1	114	Connect to GND or V _{DD} duing normal operation.
ramiesi	1	109	If ramtest = 1 lest of the on-crip RAMs is enabled.
			Set ramtest = 0 for normal operation.
pilseiect	1	178	If pilselect = 0 the on-chip phase locked loops are disabled.
			Set plisalect = 1 for normal operation.
tı	ı	180	Two clocks required by the DRAM interface during test operation
tq	i	179	Connect to GND or V _{DD} during normal operation.
pdout	0	207	These two pins are connections for an
pdin	1	206	external filter for the phase lock loop.

Table A.17.2 Temporal Decoder Test signals

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	210
nc	208	nc	156	nc	104	nc	52
test pin	207	nc	155	nc	103	ne	51
test pin	206	ira	154	nc	102	re	50
GND	205	nc	153	VD0	101	DRAM_data(15)	19
OE	204	data[7]	152	out_accept	100	nc	48
DRAM_acor(0)	203	data(6)	151	out_valid	99	DRAM_Sata(15)	47
700	202	nc	150	out_data(0)	98	ne	46
nc	201	data(5)	149	out_data(1)	97	GND	45
ORAM_addr(1)	200	nc	148	GND	96	DRAM_data[17]	11
ORAM_addr(2)	199	data[4]	147	out_data(2)	95	ne	43
GND	198	GNO	146	out_data(3)	94	DRAM_data(13)	42
ORAM_acdr(3)	197	data(3)	145	out_data[4]	93	V20	41
пс	196	nc	144	out_data(5)	92	ne	40

Table A.17.3 Temporal Decoder Pin Assignments

340

Signal Name	- Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	2
DRAM_addr(4)	195	data(2)	143	V00	91	CRAM_data(19] 39
v00	194	nc	142	out_data(6)	90	CRAM_data(20	
DPAM_addr(5)	193	data(1)	141	out_data(7)	89	l ne	37
ORAM_acor(6)	192	data(0)	140	nc	88	GND	35
nc	191	nc	139	out_extn	87	CPAM_data(2)	
GND	190	voo	138	GND	86	rc	
DRAM_addr(7)	189	addr[7]	137	DRAM_data(0)	85		34
DRAM_addr[8]	188	addr(6)	136	DRAM_data[1]	84	CRAM_data/22]	33
V00	187	addr(5)	135	DRAM_cata[2]	83	V20	32
DRAM_addr(9)	186	GND	134	T T		CPAM_data(23)	31
nc	185	addr[4]	133	DRAM_data(3)	82	DRAM_data(24)	30
DRAM_addr(10)	184	addr(3)	132	nc nc	81	re	29
GND	183	addr(2)	131	1	80	GND	28
nc	182	addr(1)	130	DRAM_data[4]	79	DRAM_cata(25)	27
V00	181			GND	78	ne	25
test pin	180	v00 addr(0)	129	nc	77	DRAM_data(25)	25
test pin	179	enable(0)	128	DRAM_data(5)	75	nc	51
test pin	178		127	nc	75	v00	23
decoder_clock	177	enable(1)	126	DRAM_data[6]	74	DPAM_data(27)	22
ne			125	voo	73	FC	21
GND	176	GND	124	DRAM_data[7]	72	DPAM_data(28)	20
	175	test pin	123	nc	71	DRAM_data(29)	1.9
n_extn	174	test pin	122	DRAM_data(8)	70	GND	18
n_data(8)	173	trst	121	GND	69	DRAM_data(30	17
n_data(7)	172	tdo	120	DRAM_data(9)	58	nc	15
n_data(6)	171	nc	119	nc	67	DRAM_data(31)	15
00	170	V00	118	DRAM_data[10]	66	voo	14
1_data(5)	169	trns	117	YD0	65	nc	13
1_data(4)	168	tdi	116	nc	54	WE	12
_cata(3)	167	tck	115	DRAM_data(11)	63	FAS	:1
_cata(2)	166	test pin	114	nc	52	nc	٠:٥
ND	165	GND	113	DRAM_cata(12)	61	GND	
_data[1]	154	DRAM_enable	112	GND	60		
_cata/0]	163	test pin	111	DRAM_sata(13)	59		
_valid	162	test pin	110	ne	58		5
_accept	161	test pin	109	DRAM_data(14)	57	-(-)	<u> </u>

Table A.17.3 Temporal Decoder Pin Assignments (contd)

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
reset	160	nc	108	voo	56	CAS(2)	4
VD0	159	nc	107	nc	55	nc	3
nc	158	nc	106	ne	54	CAS(3)	2
n¢	157	nc	105	nc	53	nc	1

Table A.17.3 Temporal Decoder Pin
Assignments (contd)

A.17.1.1 "nc" no connect pins

The pins labelled nc in Table A.17.3 are not currently used in the present invention and are reserved for future products. These pins should be left unconnected. They should not be connected to $V_{\rm DD}$, GND, each other or any other signal.

A.17.1.2 V_{DD} and GND pins

As will be appreciated all the V_{DD} and GND pins provided must be connected to the appropriate power supply. The device will not operate correctly unless all the V_{DD} and GND pins are correctly used.

A.17.1.3 Test pin connections for normal operation

Nine pins on the Temporal Decoder are reserved for internal test use.

Pin number	Connection
	Connect to GND for normal operation
	Connect to V _{DD} for normal operation
	Leave Open Circuit for normal operation

Table A.17.4 Default test pin connections

A.17.1.4 JTAG pins for normal operation See Section A.8.1.

Addr. (hex)	Register Name	See table		
0x00 0x01	Interrupt service area	A 17 6		
0x02 0x07	Not used			
0×08	Chip access	A 17 7		
0x09 0x0F	Not used			
0x10	Picture sequencing	A.17 8		
0x11 0x1F	Not used			
0×20 0×2E	DRAM interface configuration registers A 17.9			
0x2F 0x3F	Not used			
0x40 0x53	Buffer configuration A.17 8			
0x54 0x5F	Not used			
0x60 0xFF	Test registers A.17.11			

Table A.17.5 Overview of Temporal Decoder memory map

Addr.	Sit	Register Name	Page references	
(hex)	ոսու.	•		
0x00	7	chip_event		
	6:2	not used		
	1	chip_stopped_event		
	0	count_error_event	1	
0x01 7 chip_mask		chip_mask		
	6:2	not used	1	
1 chip_stopped_mask		chip_stopped_mask	1	
	0	count_error_mask		

Table A.17.6 Interrupt service area registers

Addr.	Sit	_		-
(hex)	aum,	Register Na	Page references	
0×08	7.1	not used		:
	0	chip_access		i

Table A.17.7 Chip access register

Addr.	Bit		
(hex)	חטות,	Register Name	Page references
0x10	7:1	not used	
	0	MPEG_reordering	
			1

Table A.17.8 Picture sequencing

. .

Addr.	Bit)
(hex) num		Register Name	Page reference
0x20	7:5	not used	
	4:0	page_start_length(4:0]	1
0x21	7:4	not used	
	3:0	read_cycle_length[3:0]	
0x22	7:4	not used	
	3:0	write_cycle_length(3:0]	
0x23	7:4	not used	
	3:0	refresh_cycle_lengtn(3:0)	I
0x24	7:4	not used	
	3:0	CAS_falling(3:0)	
0x25	7:4	not used	
	3:0	RAS_falling(3:0)	
0x26	7:1	not used	
	0	interface_timing_access	
0x27	7:0	not used	
0x28	7:6	RAS_strength(2:0)	
	5:3	OEWE_strength(3:0)	
	2:0	DRAM_data_strength(3:0)	
0x29	7	not used	
	6:4	DRAM_addr_strength(3:0)	
	3:1	CAS_strength[3:0]	
Γ	0	RAS_strength[3]	<u> </u>

Table A.17.9 DRAM interface configuration registers

Addr.	Br		
+ -		Register Name	Page references
(hex)	വാന്ന.	,	ago verei a cos
0×28	7	not used	
	6.4	DRAM_addr_strengtn(3:0)	
	3:1	CAS_strength(3:0)	
	0	RAS_strength[3]	
Cx29	7.6	RAS_strength(2:0)	
	5:3	OEWE_strength(3:0)	
	2:0	DRAM_data_strength[3:0]	
0×2A	7.0	refresh_interval	
0x25	7:0	not used	
0×2C	7:6	not used	
	5	DRAM_enable	
	4	no_refresh	
	3:2	row_address_bits(1:0)	
	1:0	DRAM_data_width(1:0]	ì
0x20	7:0	not used	
0×2E	7:0	Test registers	

Table A.17.9 DRAM interface configuration registers (contd)

Addr.			
(hex)		Register Name	Page references
0x40	7:0	not used	l l
0x41	0x41 7:2		ĺ
	1:0	picture_buffer_0[17:0]	
0x42	7:0		
0x43	7:0		
0x44	7:0	not used	
0x45	7:2		
	1:0	picture_buffer_1(17:0)	
0x46	7:0		
0x47	7:0	1	1

Table A.17.10 Buffer configuration registers

ſ	Addr. Bit			ĺ
-	~ Juli	J.	Register Name	Page references
	(hex)	num.		
	0x48	7:0	not used	
	0x49	7:1		
l		0	component_offset_0(16:0]	
	0x4A	7:0		
	0x48	7.0		
	0x4C	7.0	not used	
ĺ	Gx4D	7:1		
		0	component_offset_1[16:0]	
	0x4E	7:0		
	0x4F	7:0		
	0x50	7:0	not used	
	0x51	7:1		
		0	component_offset_2(16:0)	
	0×52	7:0		
	0×53	7:0		

Table A.17.10 Buffer configuration registers (contd)

Addr.	Bit	Register Name	Page references	
(hex)	កបកា.			
0x2E	7 4	PLL resistors		
i	3 0			
0x 6 0	7 6	not used		
	5 4	coding_standard[1:0]	<u> </u>	
	3 2	picture_type(1:0)		
	1	H261_filt		
	0	H261_s_f		
0x61	7 5	component_id	1	
	5 4	prediction_mode		
	3 0	max_sampling		
0x62	70	samp_h		
0x63	7 0	samp_v		

Table A.17.11 Test registers

Addr.	8:t	·	
(hex)	num.	Register Name	Page references
0x64	7 0	back_h	
0x65	7 0		
0x66	7 0	back_v	
0x67	7 0		
0x68	7 0	forw_n	
0x69	7 0		
0x6A	7 0	forw_v	
0×68	7 0		,
0x6C	7 0	width_in_mb	
0x6D	7 0		

Table A.17.11 Test registers (contd)

Table A.17.11 Test registers (contd)

In ra

SECTION A.18 Temporal Decoder Operation

A.18.1 Data input

The input data port of the Temporal Decoder is a standard Token Port with 9 bit wide data words. In most applications, this will be connected directly to the output Token Port of the Spatial Decoder. See Section A.4 for more information about the electrical behavior of this interface.

A.18.2 Automatic configuration

Parameters relating to the coded video's picture format are automatically loaded into registers within the Temporal Decoder by Tokens generated by the Spatial Decoder.

Token	Configuration performed		
CODING_STANDARD	The coding standard of the Temporal Decoder is automatically configured by the CODING_STANDARD Token. This is generated by the Spatial Decoder each time a new sequence is started. See Figure 58		
DEFINE SAMPLING	The horizontal and vertical chroma sampling information for each of the color components is automatically configured by DEFINE_SAMPLING Tokens.		
HORIZONTAL_MBS	The horizontal width of pictures in macro blocks is automatically configured by HORIZONTAL_MBS Token.		

Table A.18.1 Configuration of Temporal Decoder via Tokens

A.18.3 Manual configuration

The user must configure (via the microprocessor interface) application dependent factors.

15

20

5

A.18.3-1 When to configure

The Temporal Decoder should only be configured when no data processing is taking place. This is the default state after reset is removed. The Temporal Decoder can be stopped to allow re-configuration by writing 1 to the chip_access register. After configuration is complete, 0 should be written to chip_access.

See Section A.5.3 for details of when to configure the DRAM interface.

10 A.18.3.2 DRAM interface

The DRAM interface timing must be configured before it is possible to decode predictively coded video (e.g., H.261 or MPEG). See Section A.5, "DRAM Interface".

0.01

Register name	Sira/Dir	Reset State	Cescription
chip_access		1	Writing 1 to chip_access requests that the Temporal Decoder half
	~		operation to allow re-configuration. The Temporal Decoder will
chip_stopped_event	t	0	continue operating normally until it reaches the end of the surrent
	r#		video sequence. After reset is removed chip_access=1 = e The
chip_stopped_mask	1	0	Temporal Decoder is halted.
	۲*		When the chip stops a chip stopped event will occur. If
			chip_stopped_mask = t an interrupt will be generated
count_error_event	1	0	The Temporal Decoder has an adder that accs predictions to error
	r*		data. If there is a difference between the number of error data bytes
count_error_mask	1	0	and the number of prediction data bytes then a count error event is
	~		generated.
			If count_error_mask = 1 an interrupt will be generated and
			prediction forming will stop
picture_buffer_0	18	x	This event should only arise following a hardware error These specify the base addresses for the picture buffers
pictare_busies_o	rw		These specify the base accresses for the picture bone is
picture_buffer_1	18	\ x	
	~	-	
component_offset_0	17	x	These specify the offset from the picture buffer pointer at which
	~		each of the colour components is stored. Data with component (Dis-
component_offset_1	17	x	n is stored starting at the position indicated by
	~		component_offset_n. See A.3.5.1, "Component Identification
component_offset_2	17	x	number
	~		
MPEG_reordering	1	0	Setting this register to 1 makes the Temporal Decoder change the
	~		picture order from the non-causal MPEG picture sequence to the
			correct display order by the. See A 18.3.5
			This register should is ignored during JPEG and H.251 operator
			<u> </u>

Table A.18.2 Temporal Decoder registers

grant to the second

20

25

A.18.3-3 Numbers in picture buffer registers

The picture buffer pointers (18 bit) and the component offset (17 bit) registers specify a block (8x8 bytes) address, not a byte address.

5 A.18.3.4 Picture buffer allocation

To decode predictively coded video (either H.261 or MPEG) the Temporal Decoder must manage two picture buffers. See Section A.18.4 and A.18.4.4 for more information about how these buffers are used.

The user must ensure that there is sufficient memory above each of the picture buffer pointers (picture_buffer_0 and picture_buffer_1) to store a single picture of the required video format (without overlapping with the other picture buffer). Normally, one of the picture buffer pointers will be set to 0 (i.e., the bottom of memory) and the other will be set to point to the middle of the memory space.

A.18.3.4.1 Normal configuration for MPEG or H.261

H.261 and MPEG both use a 4:1:1 ratio between the different color components (i.e., there are 4 times as many luminance pels as there are pels in either of the chrominance components).

As documented in Section A.3.5.1, "Component Identification number", component 0 will be the luminance component and components 1 and 2 will be chrominance.

An example configuration of the component offset registers is to set component_offset_0 to 0 so that component 0 starts at the picture buffer pointer.

Similarly, component_offset_1 could be set to 4/6 of the picture buffer size and component_offset_2 could be set to 5/6 of the picture buffer size.

A.18.3.5 Picture sequence re-ordering

MPEG uses three different picture types: Intra (I),

<u>.`</u>.

10

20

Predicted (P) and Bidirectionally interpolated (B). B pictures are based on predictions from two pictures: one from the future and one from the past. The picture order is modified at the encoder so that I and P picture can be decoded from the coded date before they are required to decode B pictures.

The picture sequence must be corrected before these pictures can be displayed. The Temporal Decoder can provide this picture re-ordering (by setting register MPEG_reordering = 1). Alternatively, the user may wish to implement the picture re-ordering as part of his display interface function. Configuring the Temporal Decoder to provide picture re-ordering may reduce the video resolution that can be decoded, see Section A.18.5.

15 A.18.4 Prediction forming

The prediction forming requirements of H.261 decoding and MPEG decoding are quite different. The CODING_STANDARD Token automatically configures the Temporal Decoder to accommodate the prediction requirements of the different standards.

A.18.4.1 JPEG Operation

When configured for JPEG operation no predictions are performed since JPEG requires no temporal decoding.

A.18.4.2 H.261 Operation

In H.261, predictions are only from the picture just decoded. Motion vectors are only specified to integer pixel accuracy. The encoder can specify that a low pass filter be applied to the result of any prediction.

As each picture is decoded, it is written in to a picture buffer in the off-chip DRAM so that it can be used in decoding the next picture. Decoded pictures appear at the output of the Temporal Decoder as they are written into the off-chip DRAM.

For full details of prediction, and the arithmetic

10

15

20

operations involved, the reader is directed to the H.261 standard. The Temporal Decoder of the present invention is fully compliant with the requirements of H.261.

A.18.4.3 MPEG Operation (without re-ordering)

The operation of the Temporal Decoder changes for each of the three different MPEG picture types (I, P and B).

"I" pictures require no further decoding by the Temporal Decoder, but must be stored in a picture buffer (frame store) for later use in decoding P and B pictures.

Decoding P pictures requires forming predictions from a previously decoded P or I picture. The decoded P picture is stored in a picture buffer for use in decoding P and B pictures. MPEG allows motion vectors specified to half pixel accuracy. On-chip filters provide interpolation to support this half pixel accuracy.

B pictures can require predictions from both of the picture buffers. As with P pictures, half pixel motion vector resolution accuracy requires on chip interpolation of the picture information. B pictures are not stored in the off-chip buffers. They are merely transient.

All pictures appear at the output port of the Temporal Decoder as they are decoded. So, the picture sequence will be the same as that in the coded MPEG data (see the upper part of Figure 85).

25 For full details of prediction, and the arithmetic operations involved, the reader is directed to the proposed MPEG standard draft. These requirements are met by the Temporal Decoder of the present invention.

A.18.4.4 MPEG Operation (with re-ordering)

When configured for MPEG operation with picture reordering (MPEG_reordering = 1), the prediction forming
operations are as described above in Section A.18.4.3.
However, additional data transfers are performed to reorder the picture sequence.

10111-011

10

15

20

25

30

B picture decoding is as described in section A.18.4.3. However, I and P pictures are not output as they are decoded. Instead, they are written into the off-chip buffers (as previously described) and are read out only when a subsequent I or P picture arrives for decoding.

A.18.4.4.1 Decoder start-up characteristics

The output of the first I picture is delayed until the subsequent P (or I) picture starts to decode. This should be taken into consideration when estimating the start-up characteristics of a video decoder.

A.18.4.4.2 Decoder shut-down characteristics

The Temporal Decoder relies on subsequent P or I pictures to flush previous pictures out of its off-chip buffers (frame stores). This has consequences at the end of video sequences and when starting new video sequences. The Spatial Decoder provides facilities to create a "fake" I/P picture at the end of a video sequence to flush out the last P (or I) picture. However, this "fake" picture will be flushed out when a subsequent video sequence starts.

The Spatial Decoder provides the option to suppress this "fake" picture. This may be useful where it is known that a new video sequence will be supplied to the decoder immediately after an old sequence is finished. The first picture in this new sequence will flush out the last picture of the previous sequence.

A.18.5 Video resolution

The video resolution that the Temporal Decoder can support when decoding MPEG is limited by the memory bandwidth of its DRAM interface. For MPEG, two cases need to be considered: with and without MPEG picture reordering.

Sections A.18.5.2 and A.18.5.3 discuss the worst case requirements required by the current draft of the MPEG specification. Subsets of MPEG can be envisioned that have

lower memory bandwidth requirements. For example, using only integer resolution motion vectors or, alternatively, not using B pictures, significantly reduce the memory bandwidth requirements. Such subsets are not analyzed here.

A.18.5.1 Characteristics of DRAM interface

The number of cycles taken to transfer data across the DRAM interface depends on a number of factors:

The timing configuration of the DRAM interface to suite the DRAM employed

- The data bus width (8, 16 or 32 bits)
- · The type of data transfer:
 - ·8x8 block read or write
 - ·for prediction to half pixel accuracy
 - for prediction to integer pixel accuracy

See section A.5, "DRAM Interface", for more information about the detail configuration of the DRAM interface.

Table A.18.3 shows how many DRAM interface "cycles" are required for each type of data transfer.

Data bus width (bits)	read or write 8x8	form prediction (haif pixel accuracy)	form prediction (integer pixel accuracy)
8	1 page address + 64	4 page address + 81	4 page address - 64
	transfers	transfers	transfers
16	1 page address + 32	4 page address + 45	4 page accress - 40
	transfers	transfers	transfers
32	1 page address + 16	4 page address + 27	4 page address + 24
	transfers	transfers	translers

Table A.18.3 Data transfer times for Temporal Decoder

20

10

15

10

15

. 20

Table A.18.4 takes the figures in Table A.18.3 and evaluates them for a "typical" DRAM. In this example, a 27 MHz clock is assumed. It will be appreciated that while 27 MHz is used here, it is not intended as a limitation. The access start takes 11 ticks (102ns) and the data transfer takes 6 ticks (56 ns).

A.18.5.2 MPEG resolution without re-ordering

The peak memory bandwidth load occurs when decoding B pictures. In a "worst case" scenario, the B frame may be formed from predictions from both the picture buffers with all predictions being to half pixel accuracy.

Data bus width (bits)	read or write 8x8 block	form prediction (half pixel accuracy)	form prediction (integer pixel accuracy)
8	3657 ns	4907 ns	3963 ns
16	1880 ns	2907 ns	2185 ns
32	991 ns	1907 ns	1741 ns

Table A.18.4 Illustration with "typical" DRAM

Using the example figures from Table A.18.4, it can be seen that it wir' take the DRAM interface 3815 ns to read the data required for two accurate half pixel accurate predictions (via a 32 bit wide interface). The resolution that the Temporal Decoder can support is determined by the number of these predictions that can be performed within one picture time. In this example, the Temporal Decoder can process 8737 8x8 blocks in a single 33 ms picture period (e q., for 30 Hz video).

If the required video format is 704 x 480, then each picture contains 7920 8 x 8 blocks (taking into consideration the 4:2:0 chroma sampling). It can be seen that this video format consumes approx. 91% of the available DRAM interface bandwidth (before any other factors such as DRAM refresh are taken into consideration). Accordingly, the Temporal Decoder can support this video format.

A.18.5.3 MPEG resolution with re-ordering

When MPEG picture re-ordering is employed the worst case scenario is encountered while P pictures are being decoded. During this time, there are 3 loads on the DRAM interface:

· form predictions

write back the result .

read out the previous P or I picture

Using the example figures from Table A.18.3, we can find the time it takes for each of these tasks when a 32 bit wide interface is available. Forming the prediction takes 1907 ns/n while the read and the write each take 991 ns, a total of 3889 ns. This permits the Temporal Decoder to

total of 3889 ns. This permits the Temporal Decoder to process 8485 8 x 8 blocks in a 33 ms period.

Hence, processing 704 x 480 video will use approximately 93% of the available memory bandwidth (ignoring refresh). A.18.5.4 H.261

25 H.261 only supports two picture formats CIF (352 x 288) and QCIF (172 x 144) at picture rates up to 30 Hz. A CIF picture contains 2376 8 x 8 blocks. The only memory operations required are the writing of 8 x 8 blocks and the forming of predictions with integer accuracy motion vectors.

Using the example figures from Table A.18.4 for an 8 bit wide memory interface, it can be seen that writing each block will take 3657 ns while forming the prediction for one block will take 3963 ns/n, a total of 7620 ns per

30

block. Therefore, the processing time for a single CIF picture is about 18 ms, comfortably less than the 33 ms required to support 30 Hz video.

A.18.5.5 JPEG

The resolution of JPEG "video" that can be supported will be determined by the capabilities of the Spatial Decoder of the invention or the display interface. The Temporal Decoder does not affect JPEG resolution.

A.18.6 Events and Errors

10 A.18.6.1 Chip Stopped

In the present invention, writing 1 to chip_access requests that the Temporal Decoder halt operation to allow re-configuration. Once received, the Temporal Decoder will continue operating normally until it reaches the end of the current video sequence. Thereafter, the Temporal Decoder is halted.

When the chip halts, a chip stopped event will occur. If chip_stopped_mask=1, an interrupt will be generated.

A.18.6.2 Count Error

The Temporal Decoder, of the present invention, contains an adder that adds predictions to error data. If there is a difference between the number of error data bytes and the number of prediction data bytes, then a count error event is generated.

25 If count_error_mask = 1 an interrupt will be generated and forming prediction will stop.

Writing 1 to count_error_event clears the event and allows the Temporal Decoder to proceed. The DATA Token that caused the error will then proceed. However, the DATA Token that caused the error will not be of the correct length (64 bytes). This is likely to cause further problems. Thus, a count error should only arise if a significant hardware error has occurred.

15

25

SECTION A.19 Connecting to the output of the

Temporal Decoder

The output of the Temporal Decoder is a standard Token Port with 8 bit wide data words. See Section A.4 for more information about the electrical behavior of the interface.

The Tokens present at the output of the Temporal Decoder will depend on the coding standard employed and, in the case of MPEG, whether the pictures are being re-ordered. This section identifies which of the Tokens are available at the output of the Temporal decoder and which are the most useful when designing circuits to display that output. Other Tokens will be present, but are not needed to display the output and, therefore they are not discussed here.

This section concentrates on showing:

- · How the start and end of sequences can be identified.
- · How the start and end of pictures can be identified.
- · How to identify when to display the picture.
- · How to identify where in the display the picture data should be placed.

20 A.19.1 JPEG output

The Token sequence output by the Temporal Decoder when decoding JPEG data is identical to that seen at the output of Spatial Decoder. Recall, JPEG does not require processing by the Temporal Decoder. However, the Temporal Decoder tests intra data Tokens for negative values (resulting from the finite arithmetic precision of the IDCT in the Spatial Decoder) and replaces them with zero.

See Section A.16 for further discussion of the output sequence observed during JPEG operation.

15

20

25

30

A.19.2 H.261 Output

A.19.2.1 Start and end of sessions

H.261 doesn't signal the start and end of the video stream within the video data. Nevertheless, this is implied by the application. For example, the sequence starts when the telecommunication connection is made and ends when the line is dropped. Thus, the highest layer in the video syntax is the "picture layer".

The Start Code Detector of the Spatial Decoder in accordance with the invention, allows SEQUENCE_START and CODING_STANDARD Tokens to be inserted automatically before the first PICTURE_START. See sections A.11.7.3 and A.11.7.4.

At the end of an H.261 session (e.g., when the line is dropped) the user should insert a FLUSH Token after the end of the coded data. This has a number of effects (see Appendix A.31.1:

It ensures that PICTURE_END is generated to signal the end of the last picture.

It ensures that the end of the coded data is pushed through the decoder.

A.19.2.2 Acquiring pictures

Each picture is composed of a hierarchy of elements referred to as layers in the syntax. The sequence of Tokens at the output of the Temporal Decoder when decoding H.261 reflects this structure.

A.19.2.1 Picture layer

Each picture is preceded by a PICTURE_START Token and each is immediately followed by a PICTURE_END Token. H.261 doesn't naturally contain a picture end. This Token is inserted automatically by the Start Code Detector of the Spatial Decoder.

After the PICTURE_START Token, there will be TEMPORAL REFERENCE and PICTURE_TYPE Tokens. The

10

20

25

30

TEMPORAL_REFERENCE Token carries a 10 bit number (of which only the 5 LSBs are used in H.261) that indicates when the picture should be displayed. This should be studied by any display system as H.261 encoders can omit pictures from the sequence (to achieve lower data rates). Omission of pictures can be detected by the temporal reference incrementing by more than one between successive pictures.

Next, the PICTURE_TYPE Token carries information about the picture format. A display system may study this information to detect if CIF or QCIF pictures are being decoded. However, information about the picture format is also available by studying registers within the Huffman decoder.

<Iref to Huffman decoder section>

15 A.19.2.2.2 Group of Blocks Layer

Each H.261 picture is composed of a number of "groups of blocks". Each of these is preceded by a SLICE START Token (derived from the H.261 group number and group start code). This Token carries an 8 bit value that indicates where in the display the group of blocks should be placed. provides an opportunity for the decoder to resynchronize after data errors. Moreover, it provides the encoder with a mechanism to skip blocks if there are areas of a picture that do not require additional information in order to describe them. By the time SLICE START reaches the output of the Temporal Decoder, this information is effectively redundant as the Spatial Decoder and Temporal Decoder have already used the information to ensure that each picture contains the correct number of blocks and that they are in the correct positions. Hence, it should be possible to compute where to position a block of data output by the Temporal Decoder just by counting the number of blocks that have been output since the start of the picture.

The number carried by SLICE START is one less than the

H.261 group of blocks number (see the H.261 standard for more information). Figure 94 shows the positioning of H.261 groups of blocks within CIF and QCIF pictures. NOTE: in the present invention, the block numbering shown is the same as that carried by SLICE_START. This is different from the H.261 convention for numbering these groups.

Between the SLICE_START (which indicates the start of each group of blocks) and the first macroblock there may be other Tokens. These can be ignored as they are not required to display the picture data.

10

15

20

25

30

A.19.2.2.3 Macroblock layer

The sequence of macroblocks within each group of blocks is defined by H.261. There is no special Token information describing the position of each macroblock. The user should count through the macroblock sequence to determine where to display each piece of information.

Figure 96 shows the sequence in which macroblocks are placed in each group of blocks.

Each macroblock contains 6 DATA Tokens. The sequence of DATA Tokens in each group of 6 is defined by the H.261 macroblock structure. Each DATA Token should contain exactly 64 data bytes for an 8x8 area of pixels of a single color component. The color component is carried in a 2 bit number in the DATA Token (see section A.3.5.1). However, the sequence of the color components in H.261 is defined.

Each group of DATA Tokens is preceded by a number of Tokens communicating information about motion vectors, quantizer scale factors and so forth. These Tokens are not required to allow the pictures to be displayed and, thus, can be ignored.

Each DATA Token contains 64 data bytes for an 8x8 of a single color component. These are in a raster order.

A.19.3 MPEG output

MPEG has more layers in its syntax. These embody concepts such as a video sequence and the group of pictures.

A.19.3.1 MPEG Sequence layer

A sequence can have multiple entry points (sequence starts) but should have only a single exit point (sequence end). When an MPEG sequence header code is decoded, the Spatial Decoder generates a CODING_STANDARD Token followed by a SEQUENCE_START Token.

After the SEQUENCE_START, there will be a number of

15

20

25

30

Tokens of sequence header information that describe the video format and the like. See the draft MPEG standard for the information that is signalled in the sequence header and Table A.3.2 for information about how this data is converted into Tokens. This information describing the video format is also available in registers in the Huffman decoder.

This sequence header information may occur several times within an MPEG sequence, if that sequence has several entry points.

A.19.3.2 Group of pictures layer

An MPEG group of pictures provides a different type of "entry" point to that provided at a sequence start. The sequence header provides information about the picture/video format. Accordingly, if the decoder has no

knowledge of the video format used in a sequence, it must start at a sequence start. However, once the video format is configured into the decoder, it should be possible to start decoding at any group of pictures.

MPEG doesn't limit the number of pictures in a group. However, in many applications a group will correspond to about 0.5 seconds, as this provides a reasonable granularity of random access.

The start of a group of pictures is indicated by a GROUP_START Token. The header information provided after GROUP_START includes two useful Tokens: TIME_CODE and BROKEN CLOSED.

TIME_CODE carries a subset of the SMPTE time code information. This may be useful in synchronizing the video decoder to other signals. BROKEN_CLOSED carries the MPEG closed_gap and broken_link bits. See Section A.19.3.8 for more on the implications of random access and decoding edited video sequences.

A.19.3-3 Picture layer

The start of a new picture is indicated by the PICTURE_START Token. After this Token, there will be TEMPORAL_REFERENCE and PICTURE_TYPE Tokens. The temporary reference information may be useful if the Temporal Decoder is not configured to provide picture re-ordering. The picture type information may be useful if a display system wants to specially process B pictures at the start of an open GOP (see Section A.19.3.8).

Each picture is composed of a number of slices.

A.19.3.4 Slice layer

Section A.19.2.2.2 discusses the group of blocks used in H.261. The slice in MPEG serves a similar function. However, the slice structure is not fixed by the standard.

The 8 bit value carried by the SLICE_START Token is one less than the "slice vertical position" communicated by MPEG. See the draft MPEG standard for a description of the slice layer.

By the time SLICE_START reaches the output of the

Temporal Decoder, this information is effectively redundant since the Spatial Decoder and Temporal Decoder have already used the information to ensure that each picture contains the correct number of blocks in the correct positions.

Hence, it should be possible to compute where to position a block of data output by the Temporal Decoder just by counting the number of blocks that have been output since the start of the picture.

See section A.19.3.7 for discussion of the effects of using MPEG picture re-ordering.

30 A.19.3.5 Macroblock layer

Each macroblock contains 6 blocks. These appear at the output of the Temporal Decoder in raster order (as specified by the draft MPEG specification).

(F) 0 (R) - (8

20

25

30

A.19.3-6- Block layer

Each macroblock contains 6 DATA Tokens. The sequence of DATA Tokens in each group of 6 is defined by the draft MPEG specification (this is the same as the H.261 macroblock structure). Each DATA token should contain exactly 64 data bytes for an 8 x 8 area of pixels of a single color component. The color component is carried in a 2 bit number in the DATA Token (see A.3.5.1). However, the sequence of the color components in MPEG is defined.

Each group of DATA Tokens is preceded by a number of Tokens communicating information about motion vectors, quantizer scale factors, and so forth. These Tokens are not required to allow the pictures to be displayed and, therefore, they can be ignored.

15 A.19.3.7 Effect of MPEG picture re-ordering

As described in A.18.3.5, the Temporal Decoder can be configured to provide MPEG picture re-ordering (MPEG_reordering=1). The output of P and I pictures is delayed until the next P/I picture in the data stream starts to be decoded by the Temporal Decoder. At the output of the Temporal Decoder the DATA Tokens of the newly decoded P/I picture are replaced with DATA Tokens from the older P/I picture.

When re_ordering P/I pictures, the PICTURE_START, TEMPORAL_REFERENCE and PICTURE_TYPE Tokens of the picture are stored temporarily on-chip as the picture is written into the off-chip picture buffers. When the picture is read out for display, these stored Tokens are retrieved. Accordingly, re-ordered P/I pictures have the correct values for PICTURE_START, TEMPORAL_REFERENCE and PICTURE TYPE.

All other tokens below the picture layer are not reordered. As the re-ordered P/I picture is read-out for

25

30

5

10

display it picks up the lower level non-DATA tokens of the picture that has just been decoded. Hence, these sub-picture layer Tokens should be ignored.

A.19.3.8 Random access and edited sequences

The Spatial Decoder provides facilities to help correct video decoding of edited MPEG video data and after a random access into MPEG video data.

A.19.3.8.1 Open GOPs

A group of pictures (GOP) can start with B pictures that are predicted from a P picture in a previous GOP. This is called an "open GOP". Figure 107 illustrates this. Pictures 17 and 18 are B pictures at the start of the second GOP. If the GOP is "open", then the encoder may have encoded these two pictures using predictions from the P picture 16 and also the I picture 19. Alternatively, the encoder could have restricted itself to using predictions from only the I picture 19. In this case, the second GOP is a "closed GOP".

If a decoder starts decoding the video at the first GOP, it will have no problems when it encounters the second GOP even if that GOP is open since it will have already decoded the P picture 16. However, if the decoder makes a random access and starts decoding at the second GOP it cannot decode B17 and B18 if they depend on P16 (i.e., if the GOP is open).

If the Spatial Decoder of the present invention encounters an open GOP as the first GOP following a reset or it receives a FLUSH Token, it will assume that a random access to an open GOP has occurred. In this case, the Huffman decoder will consume the data for the B pictures in the normal way. However, it will output B pictures predicted with (0,0) motion vectors off the I picture. The result will be that pictures B17 and B18 (in the example above) will be identical to I19.

10

15

20

This behavior ensures correct maintenance of the MPEG VBV rules. Also, it ensures that B pictures exist in the output at positions within the output stream expected by the other data channels. For example, the MPEG system layer provides presentation time information relating audio data to video data. The video presentation time stamps refer to the first displayed picture in a GOP, i.e., the picture with temporal reference 0. In the example above, the first displayed picture after a random access to the second GOP is B17.

The BROKEN_CLOSED Token carries the MPEG closed_gop bit. Hence, at the output of the Temporal Decoder it is possible to determine if the B pictures output are genuine or "substitutes" have been introduced by the Spatial Decoder. Some applications may wish to take special measures when these "substitute" pictures are present.

A.19.3.8.2 Edited video "

If an application edits an MPEG video sequence, it may break the relationship between two GOPs. If the GOP after the edit is an open GOP it will no longer be possible to correctly decode the B pictures at the beginning of the GOP. The application editing the MPEG data can set the broken_link bit in the GOP after the edit to indicate to the decoder that it will not be able to decode these B pictures.

If the Spatial Decoder encounters a GOP with a broken link, the Huffman decoder will decode the data for the B pictures in the normal way. However, it will output B pictures predicted with (0,0) motion vectors off the I picture. The result will be that pictures B17 and B18 (in the example above) will be identical to I19.

The BROKEN_CLOSED Token carries the MPEG broken_link bit. Hence, at the output of the Temporal Decoder it is possible to determine if the B pictures output are genuine

30

25

or "substitutes" that have been introduced by the Spatial Decoder. Some applications may wish to take special measures when these "substitute" pictures are present.



SECTION A.20 Late Write DRAM Interface

The interface is configurable in two ways:

- The detail timing of the interface can be configured to accommodate a variety of different DRAM types
- The "width" of the DRAM interface can be configured to provide a cost/performance trade-off

Second Maria	Input/	Description	
Signal Name	Cutput	Description	
DRAM_data(31:0)	1/0	The 32 bit wide DRAM data bus. Optionally this bus can be configured to	
		be 16 or 8 bits wide.	
DRAM_addr(10:0]	0	The 22 bit wide DRAM interface address is time multiplexed over this **	
		bit wide bus.	
RAS	0	The DRAM Row Address Strobe signal	
CAS(3:0]	0	The DRAM Column Address Strobe signal. One signal is provided per	
		byte of the interface's data bus. All the CAS signals are driven	
		simultaneously.	
WE	0	The DRAM Write Enable signal	
ŌΕ	0	The DRAM Output Enable signal	
eldsne_MARC	ı	This input signal, when low, makes all the output signals on the interace	
		go high impedance and stops activity on the DRAM interface.	

Table A.20.1 DRAM interface signals

	Size/	Reset	Cescription	
Register name	Dir.	State	CESCIII (IC)	
modify_DRAM_timing	1 bit	0	This function enable register allows access to the ORAM interface	
	rw		timing configuration registers. The configuration registers should not	
			be modified while this register holds the value zero. Writing a one 12	
			this register requests access to modify the configuration registers	
			After a zero has been written to this register the DRAM interace ***	
			start to use the new values in the timing configuration registers	

	Size/	Reset		
Register name		İ	Description	
	Oir.	State		
page_start_length	5 bit	0	Specifies the length of the access start in ticks. The minimum value	
	~₩		that can be used is 4 (meaning 4 licks). O selects the maximum	
			length of 32 ticks.	
read_cycle_length	4 bit	0	Specifies the length of the fast page read cycle in ticks. The	
	rw		minimum value that can be used is 4 (meaning 4 ticks). 9 selects the	
			maximum length of 15 ticks.	
write_cycle_length	4 bit	0	Specifies the length of the fast page rate write cycle in ticks. The	
	rw		minimum value that can be used is 4 (meaning 4 ticks). 0 selects the	
			maximum length of 16 ticks.	
refresh_cycle_length	4 bit	0	Specifies the length of the refresh cycle in ticks. The minimum value	
	r ~		that can be used is 4 (meaning 4 ticks). 3 selects the maximum	
			length of 16 ticks.	
RAS_falling	4 bit	0	Specifies the number of ticks after the start of the access start that	
	rw		RAS falls. The minimum value that can be used is 4 (meaning 4	
			ticks), 0 selects the maximum length of 15 ticks.	
CAS_falling	4 bit	8	Specifies the number of ticks after the start of a read cycle, write	
	rw		cycle or access start that CAS falls. The minimum value that can be	
			used is 1 (meaning 1 tick), 0 selects the maximum length of 16 licks.	
DRAM_data_width	2 bit	0	Specifies the number of bits used on the DRAM interface data bus	
	rw		DRAM_data[31:0]. See A.20.4	
row_address_bits	2 bit	0	Specifies the number of bits used for the row address portion of the	
	rw		DRAM interface address bus. See A.20.5	
DRAM_enable	1 bit	1	Writing the value 0 in to this register forces the ORAM interface into	
· -	rw		a high impedance state:	
			0 will be read from this register if either the DRAM_enable signal is	
			low or 0 has been written to the register.	

Table A.20.2 DRAM Interface configuration registers (contd)

	Size/	Reset	0	
Register name	Dir.	State	. Description	
refresh_interval	8 5:1	0	This value specifies the interval between refresh cycles in periods of	
	rw		16 decoder_clock cycles. Values in the range 1255 can be	
			configured. The value 0 is automatically loaded after reset and	
			lorces the DRAM interface to continuously execute refresh cycles	
			until a valid refresh interval is configured. It is recommended that	
			refresh_interval should be configured only once after each reset.	
no_refresh	1 bit	0	Writing the value 1 to this register prevents execution of any refrash	
	~		cycles.	
CAS_strength	3 bit	6	These three bit registers configure the output drive strength of	
RAS_strength	w		DRAM interface signals.	
addr_strength	1		This allows the interface to be configured for various different loads.	
DRAM_data_strength				
OEWE_strength			See A.20.8	

Table A.20.2 DRAM Interface configuration registers (contd)

30

A.20. f Interface timing (ticks)

In the present invention, the DRAM interface timing is derived from a clock which is running at four times the input clock rate of the device (decoder_clock). This clock is generated by an on-chip PLL.

For brevity, periods of this high speed clock are referred to as ticks.

A.20.2 Interface operation

The interface uses of the DRAM fast page mode. Three different types of access are supported:

- · Read
- ·Write
- Refresh

Each read or write access transfers a burst of between 1 and 64 bytes at a single DRAM page address. Read and write transfers are not mixed within a single access. Each successive access is treated as a random access to a new DRAM page.

A.20.3 Access structure

Each access is composed of two parts:

- 'Access start
- ·Data transfer

Each access starts with an access start and is followed by one or more data transfer cycles. There is a read, write and refresh variant of both the access start and the data transfer cycle.

At the end of the last data transfer in an access the interface enters it's default state and remains in this state until a new access is ready to start. If a new access is ready to start when the last access finishes, then the new access will start immediately.

A.20.3.1 Access start

The access start provides the page address for the read or write transfers and establishes some initial signal

conditions. There are three different access starts:

- ·Start of read
- · Start of write
- · Start of refresh

In each case the timing of RAS and the row address is controlled by the registers RAS_falling and page_start_length. The state of OE and DRAM_data[31:0] is held from the end of the previous data transfer until RAS falls. The three different access start types are only different in how they drive OE and DRAM_data[31:0] when RAS falls. See Figure 109.

Num.	Characteristic	Min.	Max.	Unit	Notes
38	RAS precharge period set by register RAS_falling	4	16	tcx	,
39	Access start duration set by register page_start_length	4	32		
- 0	CAS precharge length set by register CAS_falling.	1	15		, 4
41	Fast page read cycle length set by the register	4	15]	!
	read_cycle_length.				
42	Fast page write cycle length set by the register	4	16		:
	write_cycle_length.				
43	WE falls one tick after CAS.				i
11	Refresh cycle length set by the register refresh_cycle.	4	15]	

Table A.20.3 Access start parameters

a. This value must be less than RAS_falling to ensure CAS before RAS refresh occurs.

A.20.3-2 Data transfer

There are three different types of data transfer cycle:

- · Fast page read cycle
- · Fast page late write cycle
- 5 Refresh cycle

A start of refresh is only followed by a single refresh cycle. A start of read (or write) can be followed by one or more fast page read (or write) cycles.

At the start of the read cycle CAS is driven high and the new column address is driven.

A late write cycle is used. **WE** is driven low one tick after **CAS**. The output data is driven one tick after the address.

As a CAS before RAS refresh cycle is initiated by the start of refresh cycle, there is no interface signal activity during a refresh cycle. The purpose of the refresh cycle is to meet the minimum RAS low period required by the DRAM.

A.20.3.3 Interface default state

The interface signals enter a default state at the end of an access:

- RAS, CAS and WE high
- \cdot data and OE remain in their previous state
- addr remains stable

25 A.20.4 Data bus width

The two bit register DRAM_data_width allows the width of the DRAM interfaces data path to be configured. This allows the DRAM cost to be minimized when working with small picture formats.

10

ORAM_data_width		
04	8 bit wide data bus on DRAM_data(31:24]°.	
1	16 bit wide data bus on DRAM_data[31:16] ^[5] .	
2	32 bit wide data bus on DRAM_data(31:0).	

Table A.20.4 Configuring DRAM_data_width

- a. Default after reset.
- b. Unused signals are held high impedance.

A.20.5 Address bits

On-chip, a 24 bit address is generated. How this address is used to form the row and column addresses depends on the width of the data bus and the number of bits selected for the row address. Some configurations do not permit all the internal address bits to be used (and) therefore, produce "hidden bits).

The row address is extracted from the middle portion of the address. This maximizes the rate at which the DRAM is naturally refreshed.

A.20.5.1 Low order column address bits

The least significant 4 to 6 bits of the column address are used to provide addresses for fast page mode transfers of up to 64 bytes. The number of address bits required to control these transfers will depend on the width of the data bus (see A.20.4).

10

A.20.5.2 Row address bits

The number of bits taken from the middle section of the 24 bit internal address to provide the row address is configured by the register row_address_bits.

row_address_bits	Width of row address		
0	9 bits		
1	10 bits		
2	11 bits		

Table A.20.5 Configuring row address bits

The width of row address used will depend on the type of DRAM used and whether the MSBs of the row address are decoded off-chip to access multiple banks of DRAM.

NOTE: The row address is extracted from the middle of the internal address. If some bits of the row address are decoded to select banks of DRAM, then all possible values of these "bank select bits" must select a bank of DRAM. Otherwise, holes will be left in the address space.

row_address_bits	rów address bits	bank select	DRAM deom
0	ORAM_addr(8.0)		256k
1	DRAM_addr(8:0)	DRAM_addr(9)	256k
	[0.8] DRAM_addr		512k
	ORAM_addr(9:0)		10244
2	DRAM_addr[8:0]	DRAM_addr(10:9)	256k
	DPAM_addr(9.0]	DRAM_addr(10)	512k
	CRAM_addr(9.0)	OPAM_addr[10]	10244
	OPAM_addr(10.0)		1 2048×
	DRAM_addr[10:0]		4096k

Table A.20.6 Selecting a value for row address bits

A.20.6 DRAM Interface enable

There are two ways to make all the output signals on the DRAM interface become high impedance. The DRAM_enable register and the DRAM_enable signal. Both the register and the signal must be at a logic 1 for the DRAM interface to operate. If either is low, then the interface is taken to high impedance and data transfers through the interface are halted.

The ability to take the DRAM interface to high impedance is provided in order to allow other devices to test or to use the DRAM controlled by the Spatial Decoder (or the Temporal Decoder) when the Spatial Decoder (or the Temporal

10

15

Decoder) is not in use. It is not intended to allow other devices to share the memory during normal operation.

A.20.7 Refresh

Unless disabled by writing to the register, no_refresh, the DRAM interface will automatically refresh the DRAM using a CAS before RAS refresh cycle at an interval determined by the register refresh interval.

The value in refresh_interval specifies the interval between refresh cycles in periods of 16 decoder_clock cycles. Values in the range 1 to 255 can be configured. The value 0 is automatically loaded after reset and forces the DRAM interface to continuously execute refresh cycles (once enabled) until a valid refresh interval is configured. It is recommended that refresh_interval should be configured only once after each reset.

A.20.8 Signal strengths

The drive strength of the outputs of the DRAM interface can be configured by the user using the 3 bit registers, CAS_strength, RAS_strength, addr_strength,

DRAM_data_strength, OEWE_strength. The MSB of this 3 bit value selects either a fast or slow edge rate. The two less significant bits configure the output for different load capacitances.

The default strength after reset is 6, configuring the outputs to take approximately 10 ns to drive signal between GND and V_{DD} if loaded with 12°F.

strength value	Orive characteristics	
0	Approx. 4 ns/V into 6 pf load	
1	Approx. 4 ns/V into 12 pf load	
2	Approx. 4 ns/V into 24 pf load	
3	Approx, 4 ns/V into 48 pf load	
4	Approx. 2 ns/V into 6 pl load	
5	Approx. 2 ns/V into 12 pl load	

strength value	Orive characteristics	
64	Approx. 2 ns/V into 24 pl load	
7	Approx. 2 ns/V into 48 pl load	

Table A.20.7 Output strength configurations a. Default after reset

When an output is configured approximately for the load it is driving, it will meet the AC electrical characteristics specified in Tables A 20 11 to Table

characteristics specified in Tables A.20.11 to Table A.20.12. When appropriately configured each output is approximately matched to it's load and, therefore, minimal overshoot will occur after a signal transition.

A.20.9 After reset

- After reset, the DRAM interface configuration registers are all reset to their default values. Most significant of these default configurations are:
 - The DRAM interface is disabled and allowed to go high impedance.
- The refresh interval is configured to the special value 0 which means execute refresh cycle continuously after the interface is re-enabled.
 - The DRAM interface is set to it's slowest configuration.
- Most DRAMs require a "pause" of between 100µs and 500us

after power is first applied, followed by a number of refresh cycles before normal operation is possible.

Immediately after reset, the DRAM interface is inactive until both the DRAM_enable signal and the DRAM_enable register are set. When these have been set, the DRAM interface will execute refresh cycles (approximately every 400 ns, depending upon the clock frequency used) until the DRAM interface is configured.

The user is responsible for ensuring that the DRAM's "pause" after power_up and for allowing sufficient time after enabling the DRAM interface to ensure that the required number of refresh cycles have occurred before data transfers are attempted.

While reset is asserted, the DRAM interface is unable to refresh the DRAM. However, the reset time required by the decoder chips is sufficiently short so that is should be possible to reset them and to then re-enable the DRAM interface before the DRAM contents decay. This may be required during debugging.

Symbol	Parameter	Min.	Max.	Units
v _{oo}	Supply voltage relative to GND	-0.5	6.5	l v
VIN	Input voltage on any pin	GND - 0.5	V ₀₀ + 0.5	V
TA	Operating temperature	-40	+85	•c
Ts	Storage temperature	-55	+150	·c

Table A.20.8 Maximum Ratings'

10

15

Symbol	Parameter	Min.	Max.	Units
Voo	Supply voltage relative to GND	4.75	5.25	ļ v
GND	Ground	0	0	V
V _{IH}	input logic '1' voltage	2.0	V ₀₀ - 0.5	V
VIL	Input logic '0' voltage	GND - 0.5	0.8	l v
T _A	Operating temperature	0	70	•C•

Table A.20.9 DC Operating conditions

a. With TBA linear ft/min transverse airflow

Symbol	Parameter	Min.	Max.	Units
Vol	Ontout logic '0' voltage		0.4	V 4
Voн	Output logic "1" voltage	2.8		V
10	Output current	± 100		μΑ 5
loz	Output off state leakage current	± 20		μА
¹ ız	Input leakage current	= 10		μΑ
100	RMS power supply current		500	mA
CIN	input capacitance		5	pF
Symbol	Parameter	Min.	Max.	Units
Cour	Output / IO capacitance		5	ρF

Table A.20.10 DC Electrical characteristics (contd)

Table A.20.10 DC Electrical characteristics

- a. AC parameters are specified using $V_{\text{OLmax}} {=}\, 0.8\, V$ as the measurement level.
- b. This is the steady state drive capability of the interface. Transient currents may be much greater.

_

A.20.10.1 AC characteristics

Num.	Parameter	Min.	Max.	Unit	Note *
45	Cycle time e.g. tPC	-2	-2	rs	1
46	Cycle time e.g. tRC	-2	-2	ı ns	[
47	High pulse e.g. tRP, tCP, tCPN	-5	-2	ns	
18	Low pulse e.g. IRAS, ICAS, ICAC, IWP, IRASP, IRASC	-11	+2	ns	
19	Cycle time e.g. tACP/tCPA	-8	+2	ns	

Table A.20.11 Differences from nominal values for a strobe

Table A.20.11 Differences from nominal values for a strobe

a. The driver strength of the signal must be configured appropriately for its load

Num.	Parameter	Min.	Мах.	Unit	Note 4
50	Strobe to strobe delay e.g. tRCD, tCSR	-3	+3	лs	
51	Low hold time e.g. tRSH, tCSH, tRWL	-13	+3	ns	
	ICWL TRAC, TOAC/OE, TCHR				
52	Strope to strope precharge e.g. tCRP,	-9	-3	ns	
	tRCS, tRCH, tRRH, tRPC				1
	CAS precharge pulse between any two	-5	+2	ns !	ļ
	CAS signals on wide DRAMs e.g. ICP, or			!	
	between AAS rising and CAS falling e.g.		-		
	tRPC	<u> </u>		<u> </u>	-

Table A.20.12 Differences from nominal values between two strobes

Table A.20.12 Differences from nominal values between two strobes

Num.	Parameter	Min.	Max.	Unit	Note *
53	Precharge before disable e.g. (RHCP/	-12	+3	ns	
	СРЯН				

Table A.20.12 Differences from nominal values between two strobes (contd)

a. The driver strength of the two signals must be configured appropriately for their loads

25

30

10

SECTION B.1 Start Code Detector

B.1.1 Overview

As previously shown in Figure 11, the Start Code Detector (SCD) is the first block on the Spatial Decoder. Its primary purpose is to detect MPEG, JPEG and H.261 start codes in the input data stream and to replace them with relevant Tokens. It also allows user access to the input data stream via the microprocessor interface, and performs preliminary formatting and "tidying up" of the token data stream. Recall, the SCD can receive either raw byte data or data already assembled in Token format.

Typically, start codes are 24, 16 and 8 bits wide for MPEG, H.261, and JPEG, respectively. The Start Code Detector takes the incoming data in bytes, either from the Microprocessor Interface (upi) or a token/byte port and shifts it through three shift registers. The first register is an 8 bit parallel in serial out, the second register is of programmable length (16 or 24 bits) and is where the start codes are detected, and the third register is 15 bits wide and is used to reformat the data into 15 bit tokens. There are also two "tag" Shift Registers (SR) running parallel with the second and third SRs. contain tags to indicate whether or not the associated bit in the data SR is good. Incoming bytes that are not part of a DATA Token and are unrecognized by the SCD, are allowed to bypass the shift registers and are output when all three shift registers are flushed (empty) and the contents output successfully. Recognized non-data tokens are used to configure the SCD, spring traps, or set flags. They also bypass the shift registers and are output unchanged.

B.1.2 Major Blocks

The hardware for the Start Code Detector consists of 10 state machines.

35 B.1.2.1 Input Circuit (scdipc.sch.iplm.M)

The input circuit has three modes of operation: token, byte and microprocessor interface. These modes allow data

15

to be input either as a raw byte stream (but still using the two-wire interface), as a token stream, or by the user via the upi. In all cases, the input circuit will always output the correct DATA Tokens by generating DATA Token headers where appropriate. Transitions to and from upi mode are synchronized to the system clocks and the upi may be forced to wait until a safe point in the data stream before gaining access. The Byte mode pin determines whether the input circuit is in token or byte mode. Furthermore, initially informing the system as to which standard is being decoded (so a CODING_STANDARD Token can be generated) can be done in any of the three modes.

B.1.2.2 Token decoder (scdipnew.sch, scdipnem.M)

This block decodes the incoming tokens and issues commands to the other blocks.

Table B.1.1. Recognized input tokens

Input Token	Command	 Comments
NULL	WAIT	NULLs are removed
DATA	NORMAL	Load next byte into first SA
CODING_STD	BYPASS	Flush shift registers, perform padding, butput and switch to bypass mode. Load CODING_STANDARD register.
FLUSH	BYPASS	Flush SRs with padding, output and switch to bypass mode.
ELSE	BYPASS	Flush SRs with padding, output and switch to
(unrecognised token)		bypass mode.

Note: A change in coding standard is passed to all blocks via the two-wire interface after the SRs are flushed. This ensures that the change from one data stream to another happens at the correct point throughout the SCD. This principle is applied throughout the presentation so that a change in the coding standard can flow through the whole chip prior to the new stream.

B.1.2.3 JPEG (scdjpeg.sch scdjpegm.M)

Start codes (Markers) in JPEG are sufficiently different that JPEG has a state machine all to itself. In the present invention, this block handles all the JPEG marker detection, length counting/checking, and removal of data. Detected JPEG markers are flagged as start codes (with v_not_t - see later text) and the command from scdipnew is overridden and forced to bypass. The operation is best described in code.

```
switch (state)
{
   case (LOOKING):
    if (input == 0xff)
   {
      state = GETVALUE; /*Found a marker*/
      remove; /*Marker gets removed*/
   }
   else
```

5

10

15

```
state = LOOKING;
break; __ ~
case (GETVALUE);
 if (input = 0xM)
   state = GETVALUE; /*Overlapping markers*/
   remove;
 }
  else if (input == 0x00)
  {
   state = LOOKING;/*Wasn't a marker*/
   insert(0xff); /*Put the 0xff back*/
 }
else
  command = BYPASS; /*override command*/
   if(lc) /* Does the marker have a length count*/
     state = GETLC0;
   else
     state = LOOKING;
break;
case (GETLC0):
  loadic0; /*Load the top length count byte*/
  state = GETLC1;
  remove;
break;
case (GETLC1)
  loadic1;
  remove;
  state = DECLC;
break;
case (DECLC):
  lent = lent - 2
```

n 19

388 A

```
break;

case (CHECKLC):

if (lcnt == 0)

state = LOOKING;/*No more to do*/

else if (lcnt < 0)

state = LOOKING;/*generate Illegal_Length_Error*/

else

state = COUNT;

break;

case (COUNT):

decrement length count until 1

if (lc <= 1)

state = LOOKING;

}
```

15

20

25

30

35

B.1.2.4 Input Shifter (scinshft.sch, scinshm.M)

The basic operation of this block is quite simple. This block takes a byte of data from the input circuit, loads the shift register and shifts it out. However, it also obeys the commands from the input decoder and handles the transitions to and from bypass mode (flushing the other SRs): On receiving a BYPASS command, the associated byte is not loaded into the shift register. Instead "rubbish" (tag = 1) is shifted out to force any data held in the other shift registers to the output. The block then waits for a "flushed" signal indicating that this "rubbish" has appeared at the token reconstructor. The input byte is then passed directly to the token reconstructor.

B.1.2.5 Start Code Detector (scdetect.sch, scdetm.M)

This block includes two shift registers which are programmable to 16 or 24 bits, start code detection logic and "valid contents" detection logic. MPEG start codes require the full 24 bits, whereas H.261 requires only 16.

In the present invention, the first SR is for data and the second carries tags which indicate whether the bits in the data SR are valid - there are no gaps or stalls (in the two-wire interface sense) in the SRs, but the bits they contain can be invalid (rubbish) whilst they are being flushed. On detection of a start code, the tag shift register bits are set in order to invalidate the contents of the detector SR.

A start code cannot be detected unless the SR contents are all valid. Non byte-aligned start codes are detected and may be flagged. Moreover, when a start code is detected, it cannot be definitely flagged until an overlapping start code has been checked for. To accomplish this function, the "value" of the detected start code (the byte following it) is shifted right through scinshift, scdetect and into scoshift. Having arrived at scoshift without the detection of another start code, it is overlapping start codes have been eliminated and it is flagged as a valid start code.

15

20

25

3.0

35

B.1.2.6 Output Shifter (scoshift.sch, scoshm.M)

The basic operation of the output shifter is to take serial data (and tags) from scdetect, pack it into 15 bit words and output them. Other functions are:

5 B.1.2.6.1 Data padding

The output consists of 15 bit words, but the input may consist of an arbitrary number of bits. In order to flush, therefore, we need to add bits to make the last word up to 15 bits. These extra bits are called padding and must be recognized and removed by the Huffman block. Padding is defined to be:

After the last data bit, a "zero" is inserted followed by sufficient "ones" to make up a 15 bit word.

The data word containing the padding is output with a low extension bit to indicate that it is the end of a data token.

B.1.2.6.2 Generation of "flushed"

In accordance with the present invention, the generation of "flushed" operation involves detecting when all SRs are flushed and signalling this to the input shifter. When the "rubbish" inserted by the input shifter reaches the end of the output shifter, and the output shifter has completed its padding, a "flushed" signal is generated. This "flushed" signal must pass through the token reconstructor before it is safe for the input shifter to enter bypass mode.

B.1.2.6.3 Flagging valid start codes

If scdetect indicates that it has found a start code, padding is performed and the current data is output. The start code value (the next byte) is shifted through the detector to eliminate overlapping start codes. If the "value" arrives at the output shifter without another start code being detected, it was not overlapped and the value is passed out with a flag v_not_t (ValueNotToken) to indicate that it is a start code value. If, however, another start code is detected (by scdetect) whilst the output shifter is waiting for the value, an overlapping start error is

15

20

25

generated. In this case, the first value is discarded and the system then waits for the second value. This value can also be overlapped, thus causing the same procedure to be repeated until a non-overlapped start code is found.

5 B.1.2.6.4 Tidying up after a start code

Having detected and output a good start code, a new DATA header is generated when data (not rubbish) starts arriving.

B.1.2.7 Data stream reconstructor (sctokrec.sch, sctokrem.M)

The Data Stream reconstructor has two-wire interface inputs: one from scinshift for bypassed tokens, and one from scoshift for packed data and start codes. Switching between the two sources is only allowed when the current token (from either source) has been completed (low extension bit arrived).

B.1.2.8 Start value to start number conversion (scdromhw.sch, schrom.M)

The process of converting start values into tokens is done in two stages. This block deals mainly with coding standard dependent issues reducing the 520 odd potential codes down to 16 coding standard independent indices.

As mentioned earlier, start values (including JPEG ones) are distinguished from all other data by a flag (value_not_token). If v_not_t is high, this block converts the 4 or 8 bit value, depending on the CODING_STANDARD, into a 4 bit start_number which is independent of the standard, and flags any unrecognized start codes.

The start numbers are as follows:

392
Table B.1.2 Start Code numbers (indices)

StarvMarker Code	Index (start_number)	Pesulting Token			
not_a_start_code	0				
sequence_start_code	1	SEQUENCE_START			
group_start_code	2	GROUP_START			
picture_start_code	3	PICTURE_START			
slice_start_code	4	SLICE_START			
user_data_start_code	5	ATAC_REZU			
extension_start_code	5	EXTENSION_DATA			
sequence_end_code	7	SEQUENCE_END			
JPEG Markers					
DHT	8	онт			
DQT	9	τρα			
DNL	10	ONL			
DRI	11	DAI			
JPEG markers that can be mapped on	JPEG markers that can be mapped onto tokens for MPEG/H.251				
SOS	picture_start_code	PICTURE_START			
SOI	sequence_start_code	SEQUENCE_5TAFT			

Table B.1.2 Start Code numbers (indices)

Start/Marker Code	Index (start_number)	Resulting Token		
EOI	sequence_end_code	SEQUENCE_END		
SOF0	group_start_code	GROUP_START		
JPEG markers that generate extn or user data				
JPG	extension_start_code	EXTENSION_DATA		
JPGn	extension_start_code	EXTENSION_DATA		
APPn	user_data_start_code	USER_DATA		
COM	user_data_start_code	USEA_DATA		
NOTE: All unrecognised JPEG markers generate an extn_start_code index				

B.1.2.9 Start number to token conversion (sconvert.sch, sconverm.M)

The second stage of the conversion is where the above start numbers (or indices) are converted into tokens. This block also handles token extensions where appropriate, discarding of extension and user data, and search modes.

20

25

Search modes are a means of entering a data stream at a random point. The search mode can be set to one of eight values:

- 0: Normal Operation find next start code.
- 5 1/2: System level searches not implemented on Spatial Decoder
 - 3: Search for Sequence or higher
 - 4: Search for group or higher
 - 5: Search for picture or higher
 - 6: Search for slice or higher
- 10 7: Search for next start code

Any non-zero search mode causes data to be discarded until the desired start code (or higher in the syntax) is detected.

This block also adds the token extensions to PICTURE and SLICE start tokens:

- ·PICTURE_START is extended with PICTURE_NUMBER, a four bit count of pictures.
- SLICE_START is extended with svp (slice vertical position). This is the "value" of the start code minus one (MPEG, H.261), and minus OXDO (JPEG).
- B.1.2.10 Data Stream Formatting (scinsert.sch, scinserx.M)

 In the present invention, Data Stream Formatting relates to conditional insertion of PICTURE_END, FLUSH,

 CODING_STANDARD, SEQUENCE_START tokens, and generation of the STOP_AFTER_PICTURE event. Its function is best simplified and described in software:

```
394
     switch (input_data)
     case (FLUSH)
      1. if (in_picture)
         output = PICTURE_END
      2. output = FLUSH
      3. if (in_picture & stop_after_picture)
        sap_error = HIGH
        in_picture = FALSE;
     4. in_picture = FALSE;
    break
    case (SEQUENCE_START)
     1. if (in_picture)
       output = PICTURE_END
    2. if (in_picture & stop_after_picture)
       2a output = FLUSH
      2b. sap_error = HIGH
        in_picture = FALSE
   3. output = CODING_STANDARD
   4. output = standard
   5. output = SEQUENCE_START
   6. in_picture = FALSE;
  break
  case (SEQUENCE_END) case (GROUP_START):
   1. if (in_picture)
     output = PICTURE_END
  2. if (in_picture & stop_after_picture) .
     2a. output = FLUSH
     2b. sap_error = HIGH
       in_picture = FALSE
  3. output = SEQUENCE_END or GROUP_START
  4. in_picture = FALSE;
break
case (PICTURE_END)
```

- 1. output = PICTURE_END
- 2. if (stop_after_picture)

2a. output = FLUSH

2b. sap_error = HIGH

3. in_picture = FALSE

break

case (PICTURE_START)

1. if (in_picture)

output = PICTURE_END

2. if (in_picture & stop_after_picture)

2a. output = FLUSH

2b. sap_error = HIGH

3. if (insert_sequence_start)

3a. output = CODING_STANDARD

3b. output = standard

3c output = SEQUENCE_START

insert_sequence_start = FALSE

4. output = PICTURE_START

in_picture = TRUE

break

default: Just pass it through

group in the

10 2

15

20

25

30

35

SECTION B.2 Huffman Decoder and Parser

B.2.1 Introduction

This section describes the Huffman Decoder and Parser circuitry in accordance with the present invention.

Figure 118 shows a high level block diagram of the Huffman Decoder and Parser. Many signals and buses are omitted from this diagram in the interests of clarity, in particular, there are several places where data is fed backwards (within the large loop that is shown).

In essence, the Huffman Decoder and Parser of the present invention consist of a number of dedicated processing blocks (shown along the bottom of the diagram) which are controlled by a programmable state machine.

Data is received from the Coded Data Buffer by the "Inshift" block. At this point, there are essentially two types of information which will be encountered: Coded data which is carried by DATA Tokens and start codes which have already been replaced by their respective Tokens by the Start Code Detector. It is possible that other Tokens will be encountered but all Tokens (other than the DATA Tokens) are treated in the same way. Tokens (start codes) are treated as a special case as the vast majority of the data will still be encoded (in H.261, JPEG or MPEG).

In the present invention, all data which is carried by the DATA Tokens is transferred to the Huffman Decoder in a serial form (bit-by-bit). This data, of course, includes many fields which are not Huffman coded, but are fixed length coded. Nevertheless, this data is still passed to the Huffman Decoder serially. In the case of Huffman encoded data, the Huffman Decoder only performs the first stage of decoding in which the actual Huffman code is replaced by an index number. If there are N district Huffman codes in the particular code table which is being decoded, then this "Huffman Index" lies in the range 0 to N-1. Furthermore, the Huffman Decoder has a "no op", i.e., "no operation" mode, which allows it to pass along data or token information to a subsequent stage without any

10

15

20

25

30

35

processing by the Huffman Decoder.

The Index to Data Unit is a relatively simple block of circuitry which performs table look-up operations. It draws its name from the second stage of the Huffman decoding process in which the index number obtained in the Huffman Decoder is converted into the actual decoded data by a simple table look-up. The Index to Data Unit cooperates with the Huffman Decoder to act as a single logical unit.

The ALU is the next block and is provided to implement other transformations on the decoded data. While the Index to Data Unit is suitable for relatively arbitrary mappings, the ALU may be used where arithmetic is more appropriate. The ALU includes a register file which it can manipulate to implement various parts of the decoding algorithms. In particular, the registers which hold vector predictions and DC predictions are included in this block. The ALU is based around a simple adder with operand selection logic. It also includes dedicated circuitry for sign-extension type operations. It is likely that a shift operation will be implemented, but this will be performed in a serial manner; there will be no barrel shifter.

The Token Formatter, in accordance with the present invention, is the last block in the Video Parser and has the task of finally assembling decoded data into Tokens which can be passed onto the rest of the decoder. At this point, there are as many Tokens as will ever be used by the decoder for this particular picture.

The Parser State Machine, which is 18 bits wide and has been adopted for use with a two-wire interface has the task of coordinating the operation of the other blocks. In essence, it is a very simple state machine and it produces a very wide "micro-code" control word which is passed to the other blocks. Figure 118 shows that the instruction word is passed from block-to-block by the side of the data. This is, indeed, the case and it is important to understand that transfers between the different blocks are controlled

20

25

30

35

by two-wire interfaces.

In the present invention, there is a two-wire interface blocks between each of the in the Video Parser. Furthermore, the Huffman Decoder works with both serial, data, the inshifter inputs data one bit at a time, and with control tokens. Accordingly, there are two modes of operation. If data is coming into the Huffman Decoder via a DATA Token, then it passes through the shifter one bit at Again, there is a two-wire interface between the inshifter and the Huffman Decoder. Other tokens, however, are not shifted in one bit at a time (serial) but rather in the header of the token. If a DATA token is input, then the header containing the address information is deleted and the data following the address is shifted in one bit at a time. If it is not a DATA Token, then the entire token, header and all, is presented to the Huffman Decoder all at once.

In the present invention, it is important to understand that the two-wire interface for the Video Parser is unusual in that it has two valid lines. One line is valid serially and one line is valid tokenly. Furthermore, both lines may not be asserted at the same time. One or the other may be asserted or if no valid data exists, then neither may be asserted although there are two valid lines, it should be recognized that there is only a single accept wire in the However, this is not a problem. other direction. Huffman Decoder knows whether it wants serial data or token information depending on what needs to be done next based Hence, the valid and accept upon the current syntax. signals are set accordingly and an Accept is sent from the Huffman Decoder to the inshifter. If the proper data or token is there, then the inshifter sends a valid signal.

For example, a typical instruction might decode a Huffman code, transform it in the Index to Data Unit, modify that result in the ALU and then this result is formed into a Token word. A single microcode instruction word is produced which contains all of the information to do this.

10

15

20

25

. 30

The command is passed directly to the Huffman Decoder which requests data bits one-by-one from the "Inshift" block until it has decoded a complete symbol. Control Tokens are input in parallel. Once this occurs, the decoded index value is passed along with the original microcode word to the Index to Data Unit. Note that the Huffman Decoder Will require several cycles to perform this operation and, indeed, the number of cycles is actually determined by the data which is decoded. The Index to Data Unit will then map this value using a table which is identified in the This value is again passed microcode instruction word. onto the next block, the ALU, along with the original microcode word. Once the ALU has completed the appropriate operation (the number of cycles may again be dependant) it passes the appropriate data onto the Token Formatting block along with the microcode word which controls the way in which the Token word is formed.

The ALU has a number of status wires or "condition codes" which are passed back to the Parser State Machine. allows the State Machine to execute conditional In fact, all instructions are conditional instructions. jump instructions; one of the conditions that may be selected is hard-wired to the value "False". By selecting this condition, a "no jump" instruction may be constructed.

In accordance with the present invention, the Token Formatter has two inputs: a data field from the ALU and/or a constant field coming from the Parser State Machine. addition, there is an instruction that tells the Token Formatter how many bits to take from one source and then to fill in with the remaining bits from the other for a total of 8 bits. For example, HORIZONTAL_SIZE has an 8 bit field invariant address identifying it HORIZONTAL_SIZE Token. In this case, the 8 bits come from the constant field and no data comes from the ALU. however, it is a DATA Token, then you would likely have 6 bits from the constant field and two lower bits indicating the color components from the ALU. Accordingly, the Token

10

15

20

25

30

Formatter takes this information and puts it into a token for use by the rest of he system. Note that the number of bits from each source in the above examples are merely for illustration purposes and one of ordinary skill in the art will appreciate that the number of bits from either source can vary.

The ALU includes a bank of counters that are used to count through the structure of the picture. The dimensions of the picture are programmed into registers associated with the counters that appear to the "microprogrammer" as part of the register bank. Several of the condition codes are outputs from this counter bank which allows conditional jumps based on "start of picture", "start of macroblock" and the like.

Note that the Parser State Machine is also referred to as the "Demultiplex State Machine". Both terms are used in this document.

Input Shifter

In the present invention, the Input Shifter is a very simple piece of circuitry consisting of a two pipeline stage datapath ("hfidp") and controlling Zcells ("hfi").

In the first pipeline stage, Token decoding takes place. At this stage, only the DATA token is recognized. Data contained in a DATA token is shifted one bit at a time into the Huffman Decoder. The second pipeline stage is the shift register. In the very last word of a DATA token, special coding takes place such that it is possible to transmit an arbitrary number of bits through the coded data buffer. The following are all possible patterns in the last data word.

15

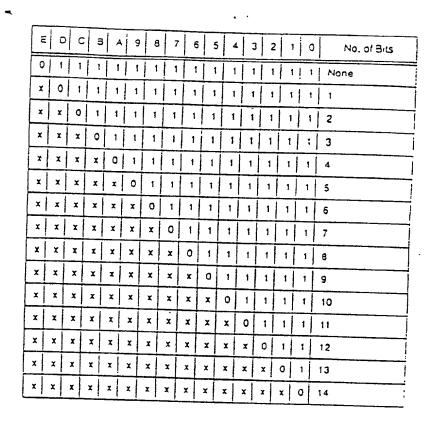


Table B.2.1 Possible Patterns in the Last Data Word

As the data bits are shifted left, one by one, in the shift register, the bit pattern "O followed by all ones" is looked for (padding). This indicates that the remaining bits in the shift register are not valid and they are discarded. Note that this action only takes place in the last word of a DATA Token.

As described previously, all other Tokens are passed to the Huffman Decoder in parallel. They are still loaded into the second pipeline stage, but no shifting takes place. Note that the DATA header is discarded and is not passed to the Huffman at all. Two "valid" wires (out valid and serial valid) are provided. Only one is asserted at a given time and it indicates what type of data is being presented at that moment.

15

20

25

30

35

B.2.2 Huffman Decoder

The Hulfman Decoder has a number of modes of operation. The most obvious is that it can decode Huffman Codes, turning them into a Huffman Index Number. In addition, it can decode fixed length codes of a length (in bits) determined by the instruction word. The Huffman Decoder can also accept Tokens from the Inshift block.

The Huffman Decode includes a very small state machine. This is used when decoding block-level information. This is because it takes too long for the Parser State Machine to make decisions (since it must wait for data to flow through the Index to Data Unit and the ALU before it can make a decision about that data and issue a new command). When this State Machine is used, the Huffman Decoder itself issues commands to the Index to Data Unit and ALU. The Huffman Decoder State Machine cannot control all of the microcode instruction bits and, therefore, it cannot issue the full range of commands to the other blocks.

B.2.2.1 Theory of Operation

When decoding Huffman codes, the Huffman Decoder of the present invention uses an arithmetic procedure to decode the incoming code into a Huffman Index Number. This number lies between 0 and N-1 (for a code table that has N entries). Bits are accepted one by one from the Input shifter.

In order to control the operation of the machine, a number of tables are required. These specify for each possible number of bits in a code (1 to 16 bits) how many codes there are of that length. As expected, this information is typically not sufficient to specify a general Huffman code. However, in MPEG, H.261 and JPEG, the Huffman codes are chosen such that this information alone can specify the Huffman Code table. There is unfortunately just one exception to this; the Tcoefficient table from H.261 which is also used in MPEG. This requires an additional table that is described elsewhere (the exception was deliberately introduced in H.261 to avoid

start code emulation).

5

10

It is important to realize that the tables used by this Huffman Decoder are precisely the same as those transmitted in JPEG. This allows these tables to be used directly while other designs of Huffman decoders would have required the generation of internal tables from the transmitted ones. This would have required extra storage and extra processing to do the conversion. Since the tables in MPEG and H.261 (with the exception noted above) can be described in the same way, a multi-standard decoder becomes practical.

The following fragment of "C" illustrates the decoding process;

```
int total = 0;
15
         int s = 0;
         int bit = 0;
         unsigned long code = 0;
         int index = 0;
         while (index>=total)
20
         if(bit>=max bits)
         fail("huff_decode: ran off end of huff table\n");
             code=(code<<1)Inext bit0;</pre>
             index=code-s+total;
25
         total += codes per bit[bit];
             s=(s+codes_per_bit[bit])<<1;
             b1t++;
         }
```

The process generally, is directly mapped into the silicon implementation although advantage is taken of the fact that certain intermediate values can be calculated in clock phases before they are required.

From the code fragment we see that;

spir swipp sm

$$=$$
 EQ 1. total_{n+1} = total_n + cpb_n

EQ 2.
$$s_{n+1} = 2(s_n + cpb_n)$$

EQ 3.
$$code_{n+1} = 2code_n + bit_n$$

EQ 4.
$$index_{a+1} = 2code_a + bit_a + total_a - 's_a$$

Unfortunately in the hardware it proved easier to use a modified set of equations in which a variable "shifted" is used in place of the variable "s". In this case;

In the hardware, however, it proved easier to use a modified set of equations in which a variable "shifted" is used in place of the variable "s". In this case;

It turns out that:

EQ 6.:
$$_{n}$$
 = 2shifted,

and so substituting this back into Equation 4 we see that:

EQ 7. index_{n+1} =
$$2(code_n - shifted_n) + total_n + bit_n$$

In addition to calculating successive values of "index", it is necessary to know when the calculation is completed. From the "C" code fragment we see that we are done when:

EQ 8.
$$index_{n+1} < total_{n+1}$$

15

20

25

Substituting from Equation 7 and Equation 1 we see that we are done when:

EQ 9. 2 (code $_a$ - shifted $_a$) + bit $_a$ - cpb $_a$ < 0

In the hardware implementation of the present invention, the common term in Equation 7 and Equation 9, $(code_n - shifted_n)$ is calculated one phase before the remainder of these equations are evaluated to give the final result and the information that the calculation is "done".

One word of warning. In various pieces of "C" code, notably the behavioral compiled code Huffman Decoder and the sm4code projects, the "C" fragment is used almost directly, but the variable "s" is actually called "shifted". Thus, there are two different variables called "shifted". One in the "C" code and the other in the hardware implementation. These two variables differ by a factor of two.

B.2.2.1.1 Inverting the Data Bits

There is one other piece of information required to correctly decode the Huffman codes. This is the polarity of the coded data. It turns out that H.261 and JPEG use opposite conventions. This reflects itself in the fact that the start codes in H.261 are zero bits whilst the marker bytes in JPEG are one bits.

In order to deal with both conventions, it is necessary to invert the coded data bits as they are read into the Huffman Decoder in order to decode H.261 style Huffman codes. This is done in the obvious manner using an exclusive OR gate. Note that the inversion is only performed for Huffman codes, as when decoding fixed length codes, the data is not inverted.

MPEG uses a mix of the two conventions. In those aspects inherited from H.261, the H.261 convention is used. In

10

15

20

30

35

those inherited from JPEG (the decoding of DC intra coefficients) the JPEG convention is used.

B.2.2.1.2 Transform Coefficients Table

When using the transform coefficients table in H.261 and MPEG, there are number of anomalies. First, the table in MPEG is a super-set of the table in H.261. In the hardware implementation of the present invention, there is no distinction drawn between the two standards and this means that an H.261 stream that contains codes from the extended part of the table (i.e., MPEG codes) will be decoded in the "correct" manner. Of course, other aspects of compression standard may well be broken. For example, these extended codes will cause start code emulation in

Second, the transform coefficient table has an anomaly that means that it is not describable in the normal manner with the codes per bit tables. This anomaly occurs with the codes of length six bits. These code words are systematically substituted by alternate code words. In an encoder, the correct result is obtained by first encoding in the normal manner. Then, for all codes that are six bits or longer, the first six bits are substituted by another six bits by a simple table look-up operation. a decoder, in accordance with the present invention, the 25 decoding process is interrupted just before the sixth bit is decoded, the code words are substituted using a table look-up, and the decoding continues.

In this case, there are only ten possible six-bit codes so the necessary look-up table is very small. operation is further helped by the fact that the upper two bits of the code are unaltered by the operation. result, it is not necessary to use a true look-up table. Instead a small collection of gates are hard-wired to give the appropriate transformation. The module that does this is called "hftcfrng". This type of code substitution is defined herein as a "ring" since each code from the set of possible codes is replaced by another code from that set

20

25

30

35

(no new codes are introduced or old codes omitted).

Furthermore, a unique implementation is used for the very first coefficient in a block. In this case, impossible for an end-of-block code to occur therefore, the table is modified so that the most commonly occurring symbol can use the code that would otherwise be interpreted as end-of-block. This may save one bit. turns out that with the architecture for decoding, in accordance with the present invention, this is easily In short, for the first bit of the first accommodated. coefficient the decoding is deemed "done" if "index" has the value zero. Furthermore, after decoding only a single bit there are only two possible values for "index", zero and one, it is only necessary to test one bit.

15 B.2.2.1.3 Register and Adder Size

The Huffman Decoder of the present invention can deal with Huffman codes that may be as long as 16 bits. However, the decoding machine is only eight bits wide. This is possible because we know that the largest possible value of the decoded Huffman Index number is 255. In fact, this could only happen in extended JPEG and, in the current application, the limit is somewhat lower (but larger than 128, so 7 bits will not suffice).

It turns out that for all legal Huffman codes, not only the final value of "index", but all intermediate values lie in the range 0 to 255. However, for an illegal code, i.e., an attempt to decode a code that is not in the current code table (probably due to a data error) the index value may exceed 255. Since we are using an eight bit machine, it is possible that at the end of decoding, the final value of "index" does not exceed 255 because the more significant bits that tell us an error has occurred have been discarded. For this reason, if at any time during decoding the index value exceeds 255 (i.e., carry out of the adder that forms index) an error occurs and decoding is abandoned.

10

15

20

25

30

35

Twelve bits of "code" are preserved. This is not necessary for decoding Huffman codes where an eight bit register would have been sufficient. These upper bits are required for fixed length codes where up to twelve bits may be read.

B.2.2.1.4 Operation for Fixed Length Codes

For fixed length codes, the "codes per bit" value is forced to zero. This means that "total" and "shifted" remain at zero throughout the operation and "index" is, therefore, the same as code. In fact, the adders and the like only allow an eight bit value to be produced for "index". Because of this, the upper bits of the output word are taken directly from the "code" register when decoding fixed length codes. When decoding Huffman codes these upper bits are forced to zero.

The fact that sufficient bits have been read from the input is calculated in the obvious manner. A comparator compares the desired number of bits with the "bit" counter.

B.2.2.2 Decoding Coefficient Data

The Parser State Machine, in accordance with the present invention, is generally only used for fairly high-level decoding. The very lowest level decoding within an eightby-eight block of data is not directly handled by this state machine. The Parser State Machine gives a command to the Huffman Decoder of the form "decode a block". Huffman Decoder, Index to Data Unit and ALU work together under the control of a dedicated state machine (essentially in the Huffman Decoder). This arrangement allows very high performance decoding of entropy coded coefficient data. There are also other feedback paths operational in this mode of operation. For instance, in JPEG decoding where the VLCs are decoded to provide SIZE and RUN information, the SIZE information is fed back directly from the output of the Index to Data Unit to the Huffman Decoder to instruct the Huffman Decoder how many FLC bits to read. addition, there are several accelerators implemented. instance, using the same example all VLC values which yield

15

20

25

30

35

a SIZE of zero are explicitly trapped by looking at the Huffman Index Value before the Index to Data stage. This means that in the case of non-zero SIZE values, the Huffman Decoder can proceed to read one FLC bit BEFORE the actual value of SIZE is known. This means that no clock cycles are wasted because this reading of the first FLC bit overlaps the single clock cycle required to perform the table look-up in the Index to Data Unit.

B.2.2.2.1 MPEG and H.261 AC Coefficient Data

10 Figure 127 shows the way in which AC Coefficients are decoded in MPEG and H.261. A flow chart detailing the operation of the Huffman Decoder is given in Figure 119.

The process starts by reading a VLC code. In the normal course of events, the Huffman index is mapped directly into values representing the six bit RUN and the absolute value of the coefficient. A one bit FLC is then read giving the sign of the coefficient. The ALU assembles the absolute value of the coefficient with this sign bit to provide the final value of the coefficient.

Note that the data format at this point is sign-magnitude and, therefore, there is little difficulty in this operation. The RUN value is passed on an auxiliary bus of six bits while the coefficients value (LEVEL) is passed on the normal data bus.

Two special cases exist and these are trapped by looking at the value of the decoded index before the Index to Data operation. These are End of Block (EOB) and Escape coded data. In the case of EOB, the fact that this occurred is passed along through the Index to Data Unit and the ALU blocks so that the Token Formatter can correctly close the open DATA Token.

Escape coded data is more complicated. First six bits of RUN are read and these are passed directly through the Index to Data Unit and are stored in the ALU. Then, one bit of FLC is read. This is the most significant bit of the eight bits of escape that are described in MPEG and H.261 and it gives the sign of the level. The sign is

10

15

20

25

35

explicitly read in this implementation because it is necessary to send different commands to the ALU for negative values versus positive values. This allows the ALU to convert the twos complement value in the bit stream into sign magnitude. In either case, the remaining seven bits of FLC are then read. If this has the value zero, then a further eight bits must be read.

In the present invention, the Huffman Decoder's internal state machine is responsible for generating commands to control itself and to also control the Index to Data Unit, the ALU and the Token Formatter. As shown in Figure 124, the Huffman Decoder's instruction comes from one of three sources, the Parser State Machine, the Huffman State Machine or an instruction stored in a register that has previously been received from the Parser State Machine. Essentially, the original instruction from the Parser State Machine (that causes the Huffman State Machine to take over control and read coefficients) is retained in a register, i.e., each time a new VLC is required, it is used. All the other instructions for the decoding are supplied by the Huffman State Machine.

B.2.2.2.2 MPEG DC Coefficient Data

This is handled in the same way as JPEG DC Coefficient Data. The same (loadable) tables are used and it is the responsibility of the controlling microprocessor to ensure that their contents are correct. The only real difference from the MPEG standard is that the predictors are reset to zero (like in JPEG) the correction for this being made in the Inverse Quantizer.

30 B.2.2.2.3 JPEG Coefficient Data

Figure 120 is a block diagram illustrating the hardware, in accordance with the present invention, for decoding JPEG AC Coefficients. Since the process for DC Coefficients is essentially a simplication of the JPEG process, the diagram serves for both AC and DC Coefficients. The only real addition to the previous diagram for the MPEG AC coefficients is that the "SSSS"

field is fed back and may be used as part of the Huffman Decoder command to specify the number of FLC bits to be read. The remainder of the command is supplied by the Huffman State Machine.

Figure 121 depicts flow charts for the Huffman decoding of both AC and DC Coefficients.

Dealing first with the process for AC Coefficients, the process starts by reading a VLC using the appropriate tables (there are two AC tables). The Huffman index is then converted into the RUN and SIZE values in the Index to Two values are trapped at the Huffman Index Data Unit. stage, these are for EOB and ZRL. These are the only two values for which no FLC bits are read. In the case when the decode index is neither of these two values, the Huffman Decoder immediately reads one bit of FLC while it waits for the Index to Data Unit to complete the look-up operation to determine how many bits are actually required. In the case of EOB, no further processing is performed by the Huffman State Machine in the Huffman Decoder and another command is read from the Parser State Machine.

In the case of ZRL, no FLC bits are required but the block is not completed. In this case, the Huffman decoder immediately commences decoding a further VLC (using the same table as before).

There is a particular problem with detecting the index values associated with ZRL and EOB. This is because (unlike H.261 and MPEG) the Huffman tables are downloadable. For each of the two JPEG AC tables, two registers are provided (one for ZRL and one for EOB). These are loaded when the table is downloaded. They hold the value of index associated with the appropriate symbol.

The ALU must convert the SIZE bit FLC code to the appropriate sign-magnitude value. These are loaded when the table is downloaded. They hold the value of index associated with the appropriate symbol.

The ALU must convert the SIZE bit FLC code to the appropriate sign-magnitude value. This can be done by

30

35

5

10

15

20

25

15

20

25

35

first sign-extending the value with the wrong sign. If the sign bit is now set, then the remaining bits are inverted (ones complement).

In the case of DC Coefficients, the decision making in the Huffman Decoding Stage is somewhat easier because there is no equivalent of the ZRL field. The only symbol which causes zero FLC bits to be read is the one indicating zero DC difference. This is again trapped at the Huffman Index stage, a register being provided to hold this index for each of the (downloadable) JPEG DC tables.

The ALU of the present invention has the job of forming the final decoded DC coefficient by retaining a copy of the last DC Coefficient value (known as the prediction). Four predictors are required, one for each of the four active color components. When the DC difference has been decoded, the ALU adds on the appropriate predictor to form the decoded value. This is stored again as the predictor for the next DC difference of that color component. Since DC coefficients are signed (because of the DC offset) conversion from twos complement to sign magnitude is required. The value is then output with a RUN of zero. fact, the instructions to perform some of the last stages of this are not supplied by the Huffman State Machine. They are simply executed by the Parser State Machine.

In a similar manner to the AC Coefficients, the ALU must first form the DC difference from the SIZE bits of FLC. However, in this case, a twos complement value is required to be added to the predictor. This can be formed by first sign extending with the wrong sign, as before. 30 result is negative, then one must be added to form the correct value. This can, of course, be added at the same time as the predictor by jamming the carry into the adder. B.2.2.3 Error Handling

Error handling deserves some mention. There effectively four sources of error that are detected:

- Ran off the end of a table.
- Serial when token expected.

15

20

25

30

35



- · Token when serial expected.
- · Too many coefficients in a block.

The first of these occurs in two situations. If the bit counter reaches sixteen (legal values being 0 to 15) then an error has occurred because the longest legal Huffman code is sixteen bits. If any intermediate value of "index" exceeds 255 then an error has occurred as described in section B.2.2.1.3.

The second occurs when serial data is encountered when a Token was expected. The third when the opposite condition arises.

The last type of error occurs if there are too many coefficients in a block. This is actually detected in the Index to Data Unit.

When any of these conditions arises, the error is noted in the Huffman error register and the Parser state machine is interrupted. It is the responsibility of the Parser State Machine to deal with the error and to issue the commands necessary to recover.

The Huffman cooperates with the Parser State Machine at the time of the interrupt in order to assure correct operation. When the Huffman Decoder interrupts the Parser State Machine, it is possible that a new command is waiting to be accepted at the output of the Parser State Machine. The Huffman Decoder will not accept this command for two whole cycles after it has interrupted the Parser State Machine. This allows the Parser State Machine to remove the command that was there (which should not now be executed) and replace it with an appropriate one. After these two cycles, the Huffman Decoder will resume normal operation and accept a command if a valid command is there. If not, then it will do nothing until the Parser State Machine presents a valid command.

When any of these errors occur, the "Huffman Error" event bit is set and, if the mask bit is set, the block will stop and the controlling microprocessor will be interrupted in the normal manner.

15

20

25

30

35

One complication occurs because in certain situations, what looks like an error, is not actually an error. most important place where this occurs is when reading the macroblock address. It is legal in the syntaxes of MPEG, H.261 and JPEG for a Token to occur in place of the expected macroblock address. If this occurs in a legal manner, the Huffman error register is loaded with zero (meaning no error) but the Parser State Machine is still interrupted. The Parser State Machine's code must this "no error" situation recognize and accordingly. In this case, the "Huffman Error" event bit will not be set and the block will not stop processing.

Several situations must be dealt with. First, the Token occurs immediately with no preceding serial bits. In this case, a "Token when serial expected error" would occur. Instead, a "no error" error occurs in the way just described.

Second, the Token is preceded by a few serial bits. In this case, a decision is made. If all of the bits preceding the Token had the value one (remember that in H.261 and MPEG the coded data is inverted so these are zero bits in the coded data file) then no error occurs. If, however, any of them were zero, then they are not valid stuffing bits and, thus, an error has occurred and a "Token when serial expected" error does occur.

Third, the token is preceded by many bits. In this case, the same decision is made. If all sixteen bits are one, then they are treated as padding bits and a "no error" error occurs. If any of them had been zero, then "Ran off Huffman Table" error occurs.

Another place that a token may occur unexpectedly is in JPEG. When dealing with either Huffman tables or Quantizer tables, any number of tables may occur in the same Marker Segment. The Huffman Decoder does not know how many there are. Because of this fact, after each table is completed it reads another 4-bit FLC assuming it to be a new table number. If, however, a new marker segment starts, then a

.... A Miller of Photograph (INCIDA)

token will be encountered in place of the 4 bit FLC. This requirement is not foreseen and, therefore, an "Ignore Errors" command bit has been added.

B.2.2.4 Huffman Commands

Here are the bits used by the Parser State Machine to control the Huffman Decoder block and their definitions. Note that the Index to Data Unit command bits are also included in this table. From the microprogrammer's point of view, the Huffman Decoder and the Index to Data Unit operate as one coherent logical block.

Sit	Name	Function
11	Ignore Errors	. Used to disable errors in certain circumstances.
10	Download	Either nominate a table for download or download data
	·	into that table.
9	Alutab	Use information from the ALU registers to specify the
		table number (or number of bits of FEC)
8	Bypass	Bypass the Index to Data Unit
7	Token	Decode a Token rather than FLC or VLC
6	First Coeff	Selects first coefficient thex for Topeff table and other
		special modes.
5	Special	. If set the Huffman State machine should take over
		control.
4	VLC (not FLC)	Specify VLC or FLC
3	Table(3)	Specify the table to use for VLC

Table B.2.2 Huffman Decoder Commands

2	Table[2]	or the number of bits to read for a FLC	
1	Table(1)	-	
0	Table(0)		:

Table B.2.2 Hufman Decoder Commands

B.2.2.4.1 Reading FLC

In this mode, Ignore Errors, Download, Alutab, Token, First Coeff, Special and VLC are all zero. Bypass will be set so that no Index to Data translation occurs.

The binary number in Table[3:0] indicates how many bits are to be read.

The numbers 0 to 12 are legal. The value zero does indeed read zero bits (as would be expected) and this instruction is, therefore, the Huffman Decoder NOP instruction. The values 13, 14 and 15 will not work and the value 15 is used when the Huffman State Machine is in control to denote the use of "SSSS" as the number of bits of FLC to read.

15 B.2.2.4.2 Reading VLC

In this mode, Ignore Errors, Download, Alutab, Token, First Coefficient and Special are zero and VLC is one. Bypass will usually be zero so that Index to Data translation occurs.

In this mode Token, First Coefficient and Special are all zero, VLC is one.

The binary number in Table[3:0] indicates which table to use as shown:

Table[3:0]	VLC Table to use
0000	TCoefficient (MPEG and H.261)
0001	CBP (Coded Block Pattern)
0010	MBA (Macroblock Address)
0011	MVD (Motion Vector Data)
0100	Intra Mtype
0101	Predicted Mtype
0110	Interpolated Mtype
0111	H.261 Mtype
10x0	JPEG (MPEG) DC Table 0
10x1	JPEG (MPEG) DC Table 1
11x0	JPEG AC Table 0
11x1	JPEG AC Table 1

Table B.2.3 Huffman Tables

Note that in the case of the tables held in RAM (i.e., the JPEG tables) bit 1 is not used so that the table selections occur twice. If a non-baseline JPEG decoder is built, then there will be four DC tables and four AC tables and Table[1] will then be required.

If Table[3] is zero, then the input data is inverted as it is used in order that the tables are read correctly as H.261 style tables. In the case of Table[3:0]=0, the appropriate Ring modification is also applied.

10

15

B.2.2.4.3 NOP Instruction

As previously described, the action of reading a FLC of zero bits is used as a No Operation instruction. No data is read from the input ports (either Token or Serial) and the Huffman Decoder outputs a data value of zero along with the instruction word.

B.2.2.4.4 TCoefficient First Coefficient

The H.261 and MPEG TCoefficient Table has a special non-Huffman code that is used for the very first coefficient in the block. In order to decode a TCoefficient at the start of a block, the First Coefficient bit may be set along with a VLC instruction with table zero. One of the many effects of the First Coefficient bit is to enable this code to be decoded.

Note that in normal operation, it is unusual to issue a "simple" command to read a TCoefficient VLC. This is because control is usually handed to the Huffman Decoder by setting the Special Bit.

B.2.2.4.5 Reading Token Words

In order to read Token words, the Token bit should be set to one. The Special and First Coefficient bits should be zero. The VLC bit should also be set if the Table[0] bit is to work correctly.

In this mode, the bits Table[1] and Table[0] are used to modify the behavior of the Token reading as follows:

Bit	Meaning	
Table(0)	Discard padding bits of senal data	
Table(1]	Discard all senal cata.	

this in Mir in

10

15

20

25

If both Table[0] and Table[1] are zero, then the presence of serial data before the token is considered to be an error and will be signalled as such.

If Table [1] is set, then all serial data is discarded until a Token Word is encountered. No error will be caused by the presence of this serial data.

If Table[0] is set, then padding bits will be discarded. It is, of course, necessary to know the polarity of the padding bits. This is determined by Table[3] in exactly the same way as for reading VLC data. If Table [3] is zero, input data is first inverted and then any "one" bits are discarded. If Table [3] is set to one, the input data is NOT inverted and "one" bits are discarded. Since the action of inverting the data depending upon the Table[3] bit is conditional on the VLC bit, this bit must be set to one. If any bits that are not padding bits are encountered (i.e., "1" bits in H.261 and MPEG) an error is reported.

Note that in these instructions only a single Token word is read. The state of the extension bit is ignored and it is the responsibility of the Demux to test this bit and act accordingly. Instructions to read multiple words are also provided - see the section on Special Instructions.

B.2.2.4.6 ALU Registers Specify Table

If the "Alutab" bit is set, registers in the ALU's register file can be used to determine the actual table number to use. The table number supplied in the command, together with the VLC bit, determines which ALU registers are used;

Table B.2.4 ALU Register Selection

VLC	table(3:0)	ALU table
0	x0xx	lwd_r_size
o	xtxx	bwd_r_size
1	x0xx	dc_huff[compid]
1	xixx	ac_huff[compid]

ирт — 11

15

20

25

35

In the case of fixed length codes, the correct number of bits are read for decoding the vectors. If r_size is zero, a NOP instruction results.

In the case of Huffman codes, the generated table number has table[3] set to one so that the resulting number refers to one of the JPEG tables.

B.2.2.4.7 Special Instructions

All of the instructions (or modes of operation) described thus far are considered as "Simple" instructions. For each command that is received, the appropriate amount of input data (of either serial of token data) is read and the resulting data is output. If no error is detected, exactly one output will be generated per command.

In the present invention, special instructions have the characteristic that more than one output word may be generated for a single command. In order to accomplish this function, the Huffman Decoder's internal State Machine takes control and will issue itself instructions as required until it decides that the instruction which the Parser requested has been complete.

In all Special instructions, the first real instruction of the sequence that is to be executed is issued with the Special bit set to one. This means that all sequences must have a unique first instruction. The advantage of this scheme is that the first real instruction of the sequence is available without a look-up operation being required based upon the command received from the Parser.

There are four recognized special instructions:

- · TCoefficient
- 30 · JPEG DC
 - . JPEG AC
 - ·Token

The first of these reads H.261 and MPEG Transform coefficients, and the like, until the end-of-block symbol is read. If the block is a non-intra block, this command will read the entire block. In this case, the "First Coefficient" bit should be set so that the first

0.000 23

15

coefficient trick is applied. If the block is an intra block, the DC term should already have been read and the "First Coefficient" bit should be zero.

In the case of an intra block in H.261, the DC term is read using a "simple" instruction to read the 8 bits FLC value. In MPEG, the "JPEG DC" special instruction described below is used.

The "JPEG DC" command is used to read a JPEG style DC term (including the SSSS bits FLC indicated by the VLC). It is also used in MPEG. The First Coefficient bit must be set in order that a counter (counting the number of coefficients) in the Index to Data Unit is reset.

The "JPEG AC" command is used to read the remainder of a block, after the DC term until either an EOB is encountered or the 64th coefficient is read.

The "Token" command is used to read an entire Token. Token words are read until the extension bit is clear. It is a convenient method of dealing with unrecognized tokens.

B.2.2.4.8 Downloading Tables.

In the present invention, the Huffman Decoder tables can be downloaded by using the "Download" bit. The first step is to nominate which table to download. This is done by issuing a command to read a FLC with both the Download and First Coeff bits set. This is treated as an NOP so no bits are actually read, but the table number is stored in a register and is used to identify which table is being

ecom in

loaded in subsequent downloading.

15

20



Table B.2.5 JPEG Tables

tab(e(3.0)	Table nominated
10xx	JPEG DC Codes per bit
IIα	JPEG AC Codes per bit
00xx	JPEG DC index to Oata
Olxx	JPEG AC Index to Data

As the above table shows, either the AC or DC tables can be loaded and table[3] determines whether it is the codesper-bit table (in the Huffman decoder itself) or the Index to Data table that is loaded.

Once the table is nominated, data is downloaded into it by issuing a command to read the required number of FLC (always 8 bits) with the Download bits set (and the First Coeff bit zero). This causes the decoded data to be written into the nominated table. An address counter is maintained, the data is written at the current address and then the address counter is incremented. The address counter is reset to zero whenever a table is nominated.

When downloading the Index to Data tables, the data and addresses are monitored. Note that the address is the Huffman Index number while the data loaded into that address is the final decoded symbol. This information is used to automatically load the registers that hold the Huffman index number for symbols of interest. Accordingly, in a JPEG AC table; when the data has the value corresponding to ZRL is recognized, the current address is written into the register CED_H_KEY_ZRL_INDEXO or CED H_KEY_ZRL_INDEXO as indicated by the table number.

Since decoded data is written into the codes-per-bit table one phase after it has been decoded, it is not possible to read data from the table during this phase.

nj. 18

Therefore, an instruction attempting to read a VLC that is issued immediately after a table download instruction will fail. There is no reason why such a sequence should occur in any real application (i.e., when doing JPEG). It is, however, possible to build simulation tests that do this.

B.2.2.5 Huffman State Machine

The Huffman State Machine, in accordance with the present invention, operates to provide the Huffman Decoder commands that are internally generated in certain cases. All of the commands that may be generated by the internal state machine may also be provided to the Huffman Decoder by the Demux.

The basic structure of the State Machine is as follows. When a command is issued to the Huffman Decoder, it is stored in a series of auxiliary latches so that it may be reused at a later time. The command is also executed by the Huffman Decoder and analyzed by the Huffman State Machine. If the command is recognized as being the first of a known instruction sequence and the SPECIAL bit is set, then the Huffman Decoder State Machine takes over control of the Huffman Decoder from the Parser State Machine.

At this point, there are three sources of instructions for the Huffman Decoder:

- 1) The Parser State Machine this choice is made at the completion of the special instruction (e.g., when EOB has been decoded) and the next demux command is accepted.
- 2) The Huffman State Machine. The Huffman State Machine may provide itself with an arbitrary command.
- 3) The original instruction that was issued by the Parser State Mchine to start the instruction.

In case (2), it is possible that the table number is provided by feedback from the Index to Data Unit, this would then replace the field in the Huffman State Machine ROM.

In case (1), in certain instances, table numbers are

25

30

3**5**

10

15

20

15

20

25

30

35

provided by values obtained from the ALU register file (e.g., in the case of AC and DC table numbers and F-numbers). These values are stored in the auxiliary command storage, so that when that command is later reused the table number is that which has been stored. It is not recovered again from the ALU since, in general, the counters will have advanced in order to refer to the next block.

Since the choice of the next instruction that will be used depends upon the data that is being decoded, it is necessary for the decision to be made very late in a cycle. Accordingly, the general structure is one in which all of the possible instructions are prepared in parallel and multiplexing late in the cycle determines the actual instruction.

Note that in each case, in addition to determining the instruction that will be used by the Huffman Decoder in the next cycle, the state machine ROM also determines the instruction that will be attached to the current data as it passes to the Index to Data Unit and then onto the ALU. In exactly the same way, all three of these instructions are prepared in parallel and then a choice is made late in the cycle.

Again, there are three choices for this part of the instruction that correspond to the three choices for the next Huffman Decoder instruction above.

- 1) A constant instruction suitable for End of Block.
- 2) The Huffman State Machine. The Huffman State Machine may provide an arbitrary instruction for the Index to Data Unit.
- 3) The original instruction that was issued by the Parser to start the instruction.

B.2.2.5.1 EOB Comparator

The EOB comparator's output essentially forces selection of the constant instruction to be presented to the Index to Data Unit and will also cause the next Huffman Instruction

15

20

30

to be the next instruction from the Parser. The exact function of the comparator is controlled by bits in the Huffman State Machine ROM.

Behind the EOB comparator, there are four registers holding the index of the EOB symbol in the AC and DC JPEG tables. In the case of the DC tables, there is of course no End-Of-Block symbol but there is the zero-size symbol, that is generated by a DC difference of zero. Since this causes zero bits of FLC to be read in exactly the same way as the EOB symbol, they are treated identically.

In addition to the four index values held in registers, the constant value, 1, can also be used. This is the index number of the EOB symbol in H.261 and MPEG.

B.2.2.5.2 ZRL Comparator

In the present invention, this is the more general purpose comparator. It causes the choice of either the Huffman State Machine instruction or the Original Instruction for use by the I to D.

Behind the ZRL comparator, there are four values. Two are in registers and hold the index of the ZRL code in the AC tables. The other two values are constants, one is the value zero and the other is 12 (the index of ESCAPE in MPEG and H.261).

The constant zero is used in the case of an FLC. The constant 12 is used whenever the table number is less than 8 (and VLC). One of the two registers is used if the table number is greater than 7 (and VLC) as determined by the low order bit of the table number.

A bit in the state machine ROM is provided to enable the Comparator and another is provided to invert its action.

If the TOKEN bit in the instruction is set, the comparator output is ignored and replaced instead by the extn bit. This allows for running until the end of a Token.

35 B.2.2.5.3 Huffman State Machine ROM

The instruction fields in the Huffman State Machine are as follows: ..

nxtstate{4:0}

The address to use in the next cycle. This address may be modified.

statectl

Allows modification of the next state address. If zero, the state machine address is unmodified, otherwise the LSB of the address is replaced by the value of either of the two comparators as follows:

nxtstate(0)	
0	Replace Lsb by EOB match
1	Replace Lsb by ZRL match

Note: in any case, if the next Huffman Instruction is selected as "Re-run original command" the state machine will jump to location 0, 1, 2 or 3 as appropriate for the command.

eobct(1:0)

This controls the selection of the next Huffman instruction based upon the EOB comparator and extn bit as follows:

eobcti[1:0]		
00	No effect - see zrictl[1:0]	
01	Take new (Parser) command if EOB	
10	Take new (Parser) command if extn low	
11	Unconditional Demux Instruction	

zrlct[1:0]

This controls the selection of the next Huffman instruction based upon the ZRL comparator. If the

10

15

20

condition is met, then it takes the state machine instruction, otherwise it re-runs the original instruction. In either case, if an eobctl*+ condition takes a demux instruction then this (eobctl*+) takes priority as follows:

zricu[1:0]	
00	Never take SM (always re-run)
01	Always take SM command
10	SM if ZAL matches
11	SM if ZRL does not match

smtab[3:0]

In the present invention, this is the table number that will be used by the Huffman Decoder if the selected instruction is the state machine instruction. However, if the ZRL comparator matches, then the zrltab[3:0] field is used in preference.

If it is not required that a different table number be used depending upon whether a ZRL match occurs, then both smtab[3:0] and zrltab[3:0] will have the same value. Note, however, that this can lead to strange simulation problems in Lsim. In the case of MPEG, there is no obvious requirement to load the registers that indicate the Huffman index number for ZRL (a JPEG only construction). However, these are still selected and the output of the ZRL comparator becomes "unknown" despite the fact that both smtab[3:0] and zrltab[3:0] have the same value in all cases that the ZRL comparator may be "unknown" (so it does not matter which is selected) the next state still goes to "unknown".

zrltab[3:0]

This is the table number that will be used by the Huffman decoder if the selected instruction is the state machine

instruction. However, if the ZRL comparator matches then the zrltab[3:0] field is used in preference.

If it is not required that a different table number be used depending upon whether a ZRL match occurs, then both smtab[3:0] and zrltab[3:0] will have the same value. Note, however, that this can lead to strange simulation problems in Lsim. In the case of MPEG, there is no obvious requirement to load the register that indicate the Huffman index number for ZRL (a JPEG only construction). However, these are still selected and the output of the ZRL comparator becomes "unknown" despite the fact that both smtab[3:0] and zrltab[3:0] have the same value in all cases that the ZRL comparator may be "unknown" (so it does not matter which is selected) the next state still goes to "unknown".

zrltab[3:0]

This is the table number that will be used by the Huffman Decoder if the selected instruction is the state machine instruction and the ZRL comparator matches.

20 smvlc

10

15

25

This is the VLC bits used by the Huffman Decoder if the selected instruction is the state machine instruction.

aluzrl(1:0)

This field controls the selection of the instruction that is passed to the ALU. It will either be the command from the Parser State Machine (that was stored at the start of the instruction sequence) or the command from the state machine:

aluzrl[1:0]	
00	Always take the saved Parser State Machine Command
01	Always take the Huffman State Machine Command
10	Take the Huffman SM command if not EOB
11	Take the Huffman SM command if not ZRL

alueob

15

20

This wire controls modification of the instruction passed to the ALU based upon the EOB comparator. This simply forces the ALU's output mode to "zinput". This is an arbitrary choice; any output mode apart from "none" will suffice. This is to ensure that the end-of-lock command word is passed to the Token Formatter block where it controls the proper formatting of DATA Tokens:

alueob	
0	Do not modify ALU outsrc field
1	Force "zinput" into outsrc if EOB match

The remainder of the fields are the ALU instruction fields. These are properly documented in the ALU description.

B.2.2.5.4 Huffman State Machine Modification

In one embodiment of the state machine, the Index to Data Unit needs to "know" when the RUN part of an escape-coded Tcoefficient is being passed to the Index to Data Unit. While this can be accomplished using an appropriate bit in the control ROM, but to avoid changing the ROM, an alternative approach has been used. In this regard, the address going into the ROM is monitored and the address value five is detected. This is the appropriate location designated in the ROM dealing with the RUN field. Of course, it will be apparent that the ROM could be programmed to use other selected address values. Moreover, the aforedescribed approach of using a bit in the control ROM could be utilized.

25 B.2.2.6 Guided Tour of Schematics

In the present invention, the Huffman Decoder is called "hd". Logically, "hd" actually includes the Index to Data Unit (this is required by the limitations of compiled code generation). Accordingly, "hd" includes the following

STREET AND MANAGEMENT AND STREET

18 7

major blocks;

Table B.2.6 Huffman Modules

Module Name	Description
hddp	Huffman Decoder (Arithmetic) dalabath
hdstdp	Huffman State Machine Datapath
hiitod	Index to Data Unit

The following description of the Huffman modules is accomplished by a global explanation of the various subsystem areas shown in greater detail in the drawings which are readily comprehended by one of ordinary skill in the art.

B.2.2.6.1 Description of "hd"

The logic for the two-wire interface control usually includes three ports controlled by the two-wire interface; data input, data output and the command. In addition, there are two "valid" wires from the input shifter; token_valid indicating that a Token is being presented on in_data[7:0] and serial_valid indicating that data is being presented on serial.

The most important signals generated are the enables that go to the latches. The most important being el which is the enable for the phl latches. The majority of pho latches are not enabled whilst two enables are provided for those that are; eo associated with serial data and eot associated with Token data.

In the present invention, the "done" signals (done, notdone and their ph0 variants done0 and notdone0) indicate when a primitive Huffman command is completed. In the case when a Huffman State Machine command is executed, "done" will be asserted at the completion of each primitive command that comprises the entire state-machine command.

25

20

10

15

10

15

20

30

35

The signal notnew prevents the acceptance of a new command from the Parser State Machine until the entire Huffman State Machine command is completed.

Regarding control of information received from the Index to Data Unit, the control logic for the "size" field is fed back to the Huffman decoder during JPEG coefficient This can actually happen in two ways. decoding. size is exactly one, this is fed back on the dedicated signal notfbone0. Otherwise, the size is fed back from the output of the Index to data unit (out_data[3:0] and a signal flvalid1 indicates that this is occurring. signal muxsize is produced to control the multiplexing of the fed-back data into the command register (sheet 10).

addition, there is feedback that exactly coefficients have been decode. Since in JPEG the EOB is not coded in this situation, the signal forceeob is produced. By analogy, with the signals for feeding back size, as mentioned above, there are in fact two ways in which this is done. Either jpegeob is used (a phl signal) or jpegeob0. Note that in the case when a normal feedback is made (jpegeob), the latch i 971 is only loaded as the data is fed back and not cleared until a new Parser State Machine command is accepted. The signal forceeob does not actually get generated until a Huffman code is decoded.

25. Thus, the fixed length code (i.e., size bits) is not affected, but the next Huffman coded information is replaced by the forced end of block. In the case when size is one and jpegeob0 is used, only one bit is read and, therefore, i 1255 and i 1256 delay the signal to the correct time. Note that it is impossible for a size of zero to occur in this situation since the only symbols with size zero are EOB and ZRL.

The decoding is fairly random decoding of the command to produce tcoeff tab0 (Huffman decoding using Tcoeff table), mba_tab0 (Huffman decoding using the MBA table) and nop (no operation). There are several reasons for generating nop. A Fixed length code of size zero is one, the forceeob

10

15

20

25

30

signal is another (since no data should be read from the input shifter even though an output is produced to signal EOB) and lastly table download nomination is a third.

notfrezero (generated by a FLC of size zero, a NOP) ensures that the result is zero when a NOP instruction is used. Furthermore, invert indicates when the serial bits should be inverted before Huffman decoding (see section B.2.2.1.1). ring indicates when the transform coefficient ring should be applied (see section B.2.2.1.2).

Decoding is also accomplished regarding addressing the codes-per-bit ROMs. These are built out of the small datapath ROMs. The signals are duplicated (e.g., csha and csla) purely to get sufficient drive by separating the ROMs into two sections. The address can be taken either from the bit counter (bit[3:0]) or from the microprocessor interface address (key-addr[3:0]) depending upon UPI access to the block being selected.

Additional decoding is concerned with the UPI reading of registers such as those that hold the Huffman index values for the JPEG tables (EOB, ZRL etc.). Also included is a tristate driver control for these registers and the UPI reading of the codes per bit RAMs.

Arithmetic datapath decoding is also provided for certain important bit numbers. first_bit is used in connection with the Tcoeff first coefficient trick and bit_five is concerned with applying the ring in the Tcoeff table. Note the use of forceeob to simulate the action that the EOB comparator matches the decoded index value.

Regarding the extn bit, if a token is read from the input shifter, then the associated extn bit is read along with it. Otherwise, the last value of extn is preserved. This allows the testing of the extn bit by the microcode program at any time after a token has been read.

When zerodat is asserted, the upper four bits of the Huffman output data are forced to zero. Since these only have valid values when decoding fixed length codes, they are zeroed when decoding a VLC, a token or when a NOP

15

20

35

instruction is executed for any reason.

Further circuitry detects when each command is completed and generates the "done" signals. Essentially, there are two groups of reasons for being "done"; normal reasons and exceptional reasons. These are each handled by one of the two three way multiplexers.

The lower multiplexer (i_1275) handles the normal reasons. In the case of a FLC, the signal ndnflc is used. This is the output of the comparator comparing the bit counter with the table number. In the case of a VLC, the signal ndnvlc is used. This is an output from the arithmetic datapath and reflects directly Equation 9. In the case of an NOP instruction or a Token, only one cycle is required and, therefore, the system is unconditionally "done".

In the present invention, the upper multiplexer (i_1274) handles exceptional cases. If the decoder is expecting a size to be fed back (fbexpctd0) in JPEG decoding and that size is one (notfbone0), then the decoder is done because only one bit is required. If the decoder is doing the first bit of the first coefficient using the Tcoeff table, it is done if bit zero of the current index is zero (see Section B.2.2.1.2). If neither of these conditions are met then there is no exceptional reason for being done.

The NOR gate (i_1293) finally resolves the "done" condition. The condition generated by i-570 (i.e., that the data is not valid) forces "done". This may seem a little strange. It is used primarily just after reset to force the machine into its "done" state in preparation for the first command ("done" resets all counters, registers, etc.). Note that any error condition also forces "done".

The signal notdonex is required for use in detecting errors. The normal "done" signals cannot be used since on detecting an error "done" is forced anyway. The use of "done" would give a combinatorial feedback loop.

Error detection and handling, is accomplished by circuitry which detects all of the possible error

10

15

20

30

35

conditions. These are 0Red together in i_1190. In this case, i_1193, i-585 and i_584 constitute the three bit Huffman error register. Note i_1253 and i-1254 which disable the error in the cases when there is no "real" error (section B.2.2.3).

In addition, i_580 and i_579 along with the associated circuitry provide a simple state machine that controls the acceptance of the first command after an error is detected.

As previously indicated, control signals are delayed to match pipeline delays in the Index to Data Unit and the ALU.

Itod_bypass is the actual bypass signal passed to the Index to Data Unit. It is modified when the Huffman State Machine is in control to force bypass whenever a fixed length code is decoded.

Aluinstr[32] is the bit that causes the ALU to feedback (condition codes) to the Parser State Machine. Furthermore, it is important when the Huffman State Machine is in control that the signals are only asserted once (rather than each time one of the primitive commands completes).

Aluinstr[36] is the bit that allows the ALU to step the block counters (if other ALU instruction bits specify an increment too). This also must only be asserted once.

In addition, these bits must only be asserted for ALU instructions that output data to the Token Formatter. Otherwise, the counters may be incremented prior to the first output to the Token formatter causing an incorrect value of "cc" in a DATA token.

In the illustrated embodiment of the invention, either alunode[1] or alunode[0] will be low if the ALU will output to the Token Formatter.

Figure 118, similar to Figure 27, illustrates the Huffman State Machine datapath referred to as "hdstdp". There is also a UPI decode for reading the output of the Huffman State machine ROM.

Multiplexing is provided to deal with the case when the

10

20

25

30

table number is specified by the ALU register file locations (see Section B.2.2.4.6).

The modification of aluinstr[3:2] deals with forcing the ALU outsrc instruction field to non-none (section B.2.2.5.3, description of alueob)

Regarding the command register for the Huffman Decoder block (x), each bit of the command has associated multiplexer which selects between the possible sources of commands. Four control signals control this selection:

Selhold causes the register to retain its current state.

Selnew causes a new command to be loaded from the Parser

State Machine. This also enables loading of the registers
that retain the original Parser State Machine command for
later use.

Selold causes loading of the command from the registers that retain the original Parser State Machine command.

/selsm causes loading of the command from the Huffman State Machine ROM.

In the case of the table number, the situation is slightly more complicated since the table number may also be loaded from the output data of the Index to Data Unit (selholdt and muxsize). Latches hold the current address in the Huffman state machine ROM. The logic detects which of the possible four commands are being executed. These signals are combined to form the lower two bits of the start address in the case of a new command.

Logic also detects when the output of the state machine ROM is meaningless (usually because the command is a "simple" command). The signal notignorerom effectively disables operation of the state machine, in particular, disabling any modification of the instruction passed to the ALU.

The circuitry generating fixstate0 controls the limited jumping capability of this state machine.

Decoding is also provided for driving the signals into the Huffman State Machine ROM. This is datapath-style combinatorial ROM.

20

The generation of escape_run is described in Section B.2.2.5.4.

Decoding also provides for the registers that hold the Huffman Index number for symbols such as ZRL and EOB. These registers can be loaded from the UPI or the datapath. The decoding in the center(es[4:0] and zs[3:0] is generating the select signals for the multiplexers that select which register or constant value to compare against the decode Huffman Index.

Regarding the control logic for the Huffman State Machine. Here the "instruction" bits from the Huffman State Machine ROM are combined with various conditions to determine what to do next and how to modify the instruction word for the ALU.

In the present invention, the signals notnew, notsm and notold are used on sheet 10 to control the operation of the Huffman Decoder command register. They are generated here in an obvious manner from the control bits in the state machine ROM (described in Section B.2.2.5.3) together with the output of the Huffman Index comparators (neobmatch and nzrlmatch).

Selection is also accomplished of the source for the instruction passed to the ALU. The actual multiplexing is performed in the Huffman State Machine datapath "hfstdp".

25. Four control signals are generated.

In the case when the end-of-block has not been encountered, one of aluseldmx (selecting the Parser State Machine instruction) or aluselsm (selecting the Huffman state machine instruction) will be generated.

In the case when the end-of-block has not been encountered, one of aluseleobd (selecting the Parser State Machine instruction) or aluseleobs (selecting the Huffman State Machine instruction) will be generated. In addition the "outsrc" field of the ALU instruction is modified to force it to "zinput".

A register holds the nominated table number during table download. Decoding is provided for the codes-per-bit RAMs.

15

20

25

35

Additional decoding recognizes when symbols like EOB and ZRL are downloaded so that the Huffman Index number registers can be automatically loaded.

Regarding the bit counter, a comparator detects when the correct number of bits have been read when reading a FLC.

B.2.2.6.2 Description of "hddp"

Comparators detect the specific values of Huffman Index. Registers hold the values for the downloadable tables. The multiplexers (meob[7:0] and mzr[7:0]) select which value to use and the exclusive-or gates and gating constitute the comparators.

Adders and registers directly evaluate the equations described in Section B.2.2.1. No further description is thought necessary here. An exclusive or is used for inverting the data (i_807) described in Section B.2.2.1.1.

The "code" register is 12 bits wide. A multiplexing arrangement implements the "ring" substitution described in Section B.2.2.1.2.

Regarding the pipeline delays for data and multiplexing between decoded serial data (index[7:0]) and Token data (ntoken0[7:0]), the Huffman index value is decided in ZRL and EOB symbols.

Codes-per-bit ROMs and their multiplexing are used for deciding which table to use. This arrangement is used because the table select information arrives late. All tables are then accessed and the correct table selected.

Regarding the codes-per-bit RAM, the final multiplexing of the codes-per-bit ROM and the output of the codes-per-bit RAM takes place inside the block "hdcpbram".

30 B.2.2.6.3 Description of "hdstdp"

In the present invention, "Hdstdp" comprises two modules. "hdstdel" is concerned with delaying the Parser State Machine control bits until the appropriate pipeline stage, e.g., when they are supplied to the ALU and Token Formatter. It only processes about half of the instruction word that is passed to the ALU, the remainder being dealt with by the other module "hdstmod".

15

20

"Hdstmod" includes the Huffman State Machine ROM. Some bits of this instruction are used by the Huffman State Machine control logic. The remaining bits are used to replace that part of the ALU instruction word (from the Parser State Machine) that is not dealt with in "hdstdel".

"Hdstmod" is obvious and requires no explanation - there are only pipeline delay registers.

"Hdstdel" is also very simple and is handled by a ROM and multiplexers for modifying the ALU instruction. The remainder of the circuitry is concerned with UPI read access to half of the Huffman State Machine ROM outputs. Buffers are also used for the control signals.

B.2.3 The Token Formatter

The Huffman Decoder Token Formatter, in accordance with the present invention, sits at the end of the Huffman block. Its function, as its name suggests, is to format the data from the Huffman Decoder into the propriety Token structure. The input data is multiplexed with data in the Microinstruction word, under control of the Microinstruction word command field. The block has two operating modes; DATA WORD, and DATA TOKEN.

B.2.3.1 The Microinstruction Word

Table B.2.7 The Microinstruction word consisting of seven fields

Field Name	Bits
Token .	0:7
Mask	8:11
Si∞k Type (Bt)	12::3
External Extn (Ee)	14
Demux Extn (De)	15
End of Block (Eb)	16
Command (Cmd)	17

17	16	15	14	12	8	<u> </u>
Omd	සා	De	Ee	ğ	Mask	Token

The Microinstruction word is governed by the same accept as the Data word.

The Microinstruction word is governed by the same accept as the Data word.

B.2.3.2 Operating Modes

Table B.2.8 Bit Allocation

Cmd	Mode
0	Data_Word
1	Data_Token

5

15

25

30

B.2.3.2.1 Data Word

In this mode, the top eight bits of the input are fed to the output. The bottom eight bits will be either the bottom eight bits of the input, the Token field of the Microinstruction word or a mixture of both, depending on the mask field. Mask represents the number of input bits in the mix, i.e.

out_data[16:8]=in_data[16:8]

out data(7:0]=(Token[7:0]&(ff<<mask))indata[7:0]

When mask is set to 0 x 8 or greater, the output data will equal the input data. This mode is used to output words in non-DATA Tokens. With mask set to 0, out_data[7:0] will be the Token field of the Microinstruction word. This mode is used for outputting Token headers that contain no data. When Token headers do contain data, the number of data bits is given by the mask field.

If External Extn(Ee) is set, out_extn=in_extn,
otherwise

out_extn=De.Bt and Eb are "don't care".

20 B.2.3.2.2 Data Token

This mode is used for formatting DATA Tokens and has two functions dependent on a signal, first_coefficient. At reset, first_coefficient is set. When the first data coefficient arrives along with a Microinstruction word that has cmd set to 1, out_data[16:2] is set to 0 x 1 and out_data[1:0] takes the value of the Bt field in the Microinstruction word. This is the header of a DATA Token. When this word has been accepted, the coefficient that accompanied the command is loaded into a register, RL and first_coefficient takes the value of Eb. When the next coefficient arrives, out_data[16:0] takes the previous coefficient, stored in RL. RL and first_coefficient are then updated. This ensures that when the end of the block is encountered and Eb is set, first_coefficient is set,

35 ready for the next DATA Token, i.e.,

```
if(first_coefficient)

out_data[16:2] = 0x1

out_data[1:0] = 3t[1:0]

RL[16:0] = in_data[16:0]

else

(
   out_data[16:0] = RL[16:0]

RL[16:0] = in_data[16:0]

Out_extn = -Eb
```

B.2.3.3 Explanatory Discussion

In accordance with the present invention, most of the instruction bits are supplied in the normal manner by the Parser State Machine. However, two of the fields are actually supplied by other circuitry. The "Bt" field mentioned above is connected directly to an output of the ALU block. This two bit field gives the current value of "cc" or "color component". Thus, when a DATA Token header is constructed, the lowest order two bits take the color component directly from the ALU counters. Secondly, the "Eb" bit is asserted in the Huffman decoder whenever and End-of-block symbols id decoded (or in the case of JPEG when one is assumed because the last coefficient in the block is coded).

The in_extn signal is derived in the Huffman Decoder. It only has meaning with respect to Tokens when the extension bit is supplied along with the Token word in the normal way.

15

20

25

B.2.4 The Parser State Machine

The Parser State Machine of the present invention is actually a very simple piece of circuitry. The complication lies in the programming of the microcode ROM which is discussed in Section B.2.5.

Essentially the machine consists of a register which holds the current address. This address is looked up in the microcode ROM to produce the microcode word. The address is also incremented in a simple incrementer and this incremented address is one of two possible addresses to be used for the next state. The other address is a field in the microcode ROM itself. Thus, each instruction is potentially a jump instruction and may jump to a location specified in the program. If the jump is not taken, control passes to the next location in the ROM.

A series sixteen condition code bits are provided. Any one of these conditions may be selected (by a field in the microcode ROM) and, in addition, it may be inverted (again a bit in the microcode ROM). The resulting signal selects between either the incremented address or the jump address in the microcode ROM. One of the conditions is hard-wired to evaluate as "False". If this condition is selected, no jump will occur. Alternatively, if this condition is selected and then inverted, the jump is always taken; an unconditional jump.



Table B.2.9 Condition Code Bits

Bit No.	Name	Description
0	user(3)	Connected to a register programmable by the user from
1	user(t)	the microprocessor interface. They allow fuser defined
2	cob_eiGu(condition codes that can be lested with little overnead.
3	CDD_SDEC:al	Two are defined to control non-standard "Coded block Pattern" processing for experimental 4 block and 3 block
		macroblock structures.
4	he[0]	These bits connect directly to the Huffman decoder's
5	he[1]	Huffman Error register.
6	he[2]	
7	Extn	The Extension bit (for Tokens)
8	Bikota	The Block Pattern Shifter
9	MBstart	At Start of a Macroblock
10	Picstart	At Start of a Picture
11	Restart	At Start of a Restart Interval
12	Chngdet	The "Sticky" Change Detect bit
13	Zero	ALU zero condition
14	Sign	ALU sign condition
15	False	Hard wired to False.

B.2.4.1 Two wire Interface Control

The two-wire interface control, in accordance with the invention, is a little unusual in this block. There is a two-wire interface between the Parser State Machine and the Huffman Decoder. This is used to control the progress of commands. The Parser State Machine will wait until a given command has been accepted before it proceeds to read the next command from the ROM. In addition, condition codes are fed back through a wire from the ALU.

Each command has a bit in the microcode ROM that allows it to specify that it should wait for feedback. If this occurs, then after that instruction has been accepted by the Huffman Decoder, no new commands are presented until

The state of the s į 153 133

10

15

20

25

30

the feedback wire from the ALU becomes asserted. This wire, fb_valid, indicates that the condition codes currently being supplied by the ALU are valid in the sense that they reflect the data associated with the command that requested the wait for feedback.

The intended use of the feature, in accordance with the present invention, is in constructing conditional jump commands that decide the next state to jump to as a result of decoding (or processing) a particular piece of data. Without this facility it would be impossible to test any conditions depending upon data in the pipeline since the two-wire control means that the time at which a certain command reaches a given processing block (i.e., the ALU in this case) is uncertain.

Not all instructions are passed to the Huffman Decoder. Some instructions may be executed without the need for the data pipeline. These tend to be jump instructions. A bit in the microcode ROM selects whether or not the instruction will be presented to the Huffman Decoder. If not, there is no requirement that the Huffman Decoder accept the instruction and, therefore, execution can continue in these circumstances even if the pipeline is stalled.

B.2.4.2 Event Handling

There are two event bits located in the Parser State Machine. One is referred to as the Huffman event and the other is referred to as the Parser Event.

The Parser Event is the simplest of these. The "condition" being monitored by this event is simply a bit in the microcode ROM. Thus, an instruction may cause a Parser Event by setting this bit. Typically, the instruction that does this will write an appropriate constant into the rom_control register so that the interrupt service routine can determine the cause of the interrupt.

After servicing a Parser Event (or immediately if the event is masked out) control resumes at the point where it left off. If the instruction that caused the event has a

10

15

20

25

30

35

jump instruction (whose condition evaluates true) then the jump is taken in the normal manner. Hence, it is possible to jump to an error handler after servicing by coding the jump.

A Huffman event is rather different. The condition being monitored is the "OR" of the three Huffman Error bits. In reality, this condition is handled in a very similar manner to the Parser Event. However, an additional wire from the Huffman Decoder, huffintrpt, is asserted whenever an error occurs. This causes control to jump to an error handler in the microcode program.

When a Huffman error occurs, therefore, the sequence involves generating interrupt and stopping the block. After servicing, control is transferred to the error handler. There is no "call" mechanism and unlike a normal interrupt, it is not possible to return to the point in the microcode before the error occurred following error handling.

It is possible for huffintrpt to be asserted without a Huffman error being generated. This occurs in the special case of a "no-error" error as discussed in Section B.2.2.3. In this case, no interrupt (to the microprocessor interface) is generated, but control is still passed to the error handler (in the microcode). Since the Huffman error register will be clear in this case, the microcode error handler can determine that this is the situation and respond accordingly.

B.2.4.3 Special locations

There are several special locations in the microcode ROM. The first four locations in the ROM are entry points to the main program. Control passes to one of these four locations on reset. The location jumped to depends upon the coding standard selected in the ALU register, coding_std. Since this location is itself reset to zero by a true reset control passes to location zero. However, it is possible to reset the Parser State Machine alone by using the UPI register bit CED_H_TRACE RST in CED H_TRACE.

30

35

In this case, the coding std register is not reset and control passes to the appropriate one of the first four locations.

The second four locations (0 x 004 to 0 x 007) are used when a Huffman interrupt takes place. Typically, a jump to the actual error handler is placed in each of these locations. Again, the choice of location is made as a result of the coding standard.

B.2.4.4 Tracing

10 As a diagnostic aid, a trace mechanism is implemented. This allows the microcode to be single-stepped. CED_H_TRACE_EVENT and CED H TRACE MASK in the register CED H TRACE control this. As their names suggest, they operate in a very similar fashion to the normal event bits. However, because of several differences (in particular no 15 UPI interrupt is ever generated) they are not grouped with the other event bits.

The tracing mechanism is turned on when CED_H_TRACE_MASK is set to one. After each microcode instruction is read from the ROM, but before it is presented to the Huffman Decoder, a trace event occurs. In this case, CED_H_TRACE_EVENT becomes one. It must be polled because no interrupt will be generated. The entire microcode word is available in the registers CED_H_KEY_DMX_WORD_0 through 25 CED_H_KEY_DMX_WORD_9. The instruction can be modified at this time if required. Writing a one to CED H TRACE EVENT the instruction to be causes executed and CED H TRACE EVENT. Shortly after this time, when the next microcode word to be executed has been read from the ROM, a new trace event will occur.

B.2.5 The Microcode

The microcode is programmed using an assembler "hpp" which is a very simple tool and much of the abstraction is achieved by using a macro preprocessor. A standard "C" preprocessor "cpp" may be used for this purpose.

The code is instructed as follows:

Ucode.u is the main file. First, this includes tokens.h

15

20

25

30

to define the tokens. Next, regfile.h defines the ALU register map. The fields.u defines the various fields in the microcode word, giving a list of defined symbols for each possible bit pattern in the field. Next, the labels that are used in the code are defined. After this step, instr.u is included to define a large number of "cpp" macros which define the basic instructions. Then, errors.h defines the numbers which define the Parser events. Next, unword.u defines the order in which the fields are placed to build the microcode word.

The remainder of ucode.u is the microcode program itself.

B.2.5.1 The Instructions

In this section the various instructions defined in ucode.u are described. Not all instructions are described here since in many cases they are small variations on a theme (particularly the ALU instructions).

B.2.5.1.1 Huffman and Index to Data Instructions

In the invention, the H_NOP instruction is used by the Huffman Decoder. It is the No-operation instruction. The Huffman does nothing in the sense that no data is decoded. The data produced by this instruction is always zero. Accordingly, the associated instruction is passed onto the ALU.

The next instructions are the Token groups; H_TOKSRCH, H_TOKSKIP_PAD, H_TOKSKIP_JPAD, H_TOKPASS and H_TOKREAD. These all read a token or tokens from the Input Shifter and pass them onto the rest of the machine. H_TOKREAD reads a single token word. H_TOKPASS can be used to read an entire token, up to and including, the word with a zero extn bit. The associated command is repeated for each word of the Token. H_TOKSRCH discards all serial data preceding a Token and then reads one token word. H_TOKSKIP_PAD skips any padding bits (H.261 and MPEG) and then reads one Token word. H_TOKSKIP_JPAD does the same thing for JPEG padding. H_FLC(NB) reads a fixed length code of "NB" bits.

 $H_{VLC}(TBL)$ reads a vic using the indicated table (passed as mnemonic, e.g., $H_{VLC}(tcoeff)$).

35

25

H_FLC_IE(NB) is like H_FLC, but the "ignore errors" bit
is set.

H_TEST_VLC(TBL) is like H_VLC, but the bypass bit is set so that the Huffman Index is passed through the Index to Data Unit unmodified.

H_FWD_R and H_BWD_R read a FLC of the size indicated by the ALU registers r_fwd_r_size and r_bwd_r_size, respectively.

H_DCJ reads JPEG style DC coefficients, the table number
10 from the ALU.

H DCH reads a H.261 DC term.

H_TCOEFF and H_DCTCOEFF read transform coefficients. In H_DCTCOEFF, the first coeff bit is set and is for non-intra blocks, whilst H_TCOEFF is for intra blocks after the DC term has already been read.

H_NOMINATE(TBL) nominates a table for subsequent
download.

 $H_DNL(NB)$ reads NB bits and downloads them into the nominated table.

20 B.2.5.1.2 ALU Instructions

There really are too many ALU instructions to explain them all in detail. The basic way in which the Mnemonics are constructed is discussed and this should make the instructions readable. Furthermore, these should readily be understandable to one of ordinary skill in the art.

Most of the ALU instructions are concerned with moving data from place to place and, therefore, a generic "load" instruction is used. In the Mnemonic, A_LDxy, it is understood that the contents of y are loaded into x., i.e.,

30 the destination is listed first and the source second:

Table B.2.10 Letters used to denote possible sources and destinations of data

Letter	Meaning			
A	A register			
Я	Aun register			
1	Data Input			
0	Data Output			
F	ALU register File			
С	Constant			
Z	Constant of zero			

By way of example, LDAI loads the A register with the data from the data input port of the ALU. If the ALU register file is specified, the mnemonic will take an address so that LDAF(RA) loads A with the contents of location RA in the register file.

The ALU has the ability to modify data as it is moved from source to destination. In this case, the arithmetic is indicated as part of the source data. Accordingly, the Mnemonic LDA_AADDF(RA) loads A with the existing contents of the A register plus the contents of the indicated location in the register file. Another example is LDA_ISGXR, which takes the input data, sign extends from the bit indicated in the RUN register, and stores the result in the A register.

In many cases, more than one destination for the same result is specified. Again, by way of example, LDF_LDA_ASUBC(RA) which loads the result of A minus a constant into both the A register and the register file.

Other mnemonics exist for specific actions. For example, "CLRA" is used for clearing the A register, "RMBC" to reset

20

15

25

30

35

the macroblock counter. These are fairly obvious and are described in comments in instr.u.

One anomaly is the use of a suffix "_O" to indicate that the result of the operation is output to the Token formatter in addition to the normal action. Thus LDFI_O(RA) stores the input data and also passes it to the token formatter. Alternatively, this could have been LDF LDO I(RA) if desired.

B.2.5.1.3 Token Formatter Instructions

This is the T_NOP "No-operation" instruction. This is really a misnomer as it is impossible to construct a no-operation instruction. However, this is used whenever the instruction is of no consequence because the ALU does not output to the Token Formatter.

15 T-TOK output a Token word.

 T_DAT output a DATA Token word (used only with the Huffman State Machine instructions).

T-GENT8 generates a token word based on the 8 bits of constant field.

T_GENT8E like T GENT8, but the extension bit is one.

 $T_{\scriptsize OPD(NB)}$ NB bits of data from the bottom NB bits of the output with the remainder of the bits coming from the constant field.

T_OPDE(NB) like T_OPD, but the extension bit is high.

T_OPD8 short-hand for T OPD(8)

T_OPD8E short-hand for T OPDE(8)

B.2.5.1.4 Parser State Machine Instructions

This instruction, D_NOP No-operation, i.e., the address increments as normal and the Parser State Machine does nothing special. The Remainder of the instruction is passed to the data pipeline. No waiting occurs.

D_WAIT is like D NOP, but waits for feedback to occur.

The simple jump group. Mnemonics like $D_{JMP}(ADDR)$ and $D_{JNX}(ADDR)$ jump if the condition is met. The instruction is not output to the Huffman Decoder.

The external jump group. Mnemonics like D_XJMP(ADDR) and D_XJNX(ADDR). These are like their simple counterparts

n (n nj) (n

above, but the instruction is output to the Huffman Decoder.

The jump and wait group. Mnemonics like $D_WJNZ(ADDR)$. These instructions are output to the Huffman Decoder and the Parser waits for feedback from the ALU before evaluating the condition.

The following Mnemonics are used for the conditions themselves.

Table B.2.11 Mnemonics used for the conditions

	Mnemonic	Meaning	
JMP		Unconditional jump	
TXL	NX	Jump if extn=1 (extn=0)	
JHE0	JNHEO	Jump if Huffman error bit 0 set (clear)	
JHE1	JNHE1	Jump if Huffman error bit 1 set (clear)	
JHE2	JNHE2	Jump if Huffman error bit 2 set (clear)	
JPTN	•	Jump if pattern shifter LSB is set	
JPICST	JNPICST	Jump is at picture start (not at picture start)	
CASTST	JNASTST	Jump if at start of restart interval (not at start)	
•	JNCPBS	Jump if not special CPB coding	
•	JNCPB8	Jump if not 8 block (i.e. 4 block) macroblock	
ıMı	JPL	Jump if negative (jump if plus)	
JZE	ZNZ	Jump il zero (jump il non-zero)	
CHNG	JNCHNG	Jump if change detect bit set (clear)	
IMBST	JNMBST	Jump if at start of macroblock (not at start)	

D_EVENT causes generation of an event.

D_DFLT for construction of a default instruction. This causes an event and then jumps to a location with the label "dflt". This instruction should never be executed since they are used to fill a ROM so that a jump to an unused location is trapped.

D_ERROR causes an event and then jumps to a label "srch_dispatch" which is assumed to attempt recovery from the error.

10

15

20

25

SECTION B.3 HUFFMAN DECODER ALU

B.3.1 Introduction

The Huffman Decoder ALU sub-block, in accordance with the present invention, provides general arithmetic and logical functionality for the Huffman Decoder block. It has the ability to do add and subtract operations, various types of sign-extend operations, and formatting of the input data into run-sign-level triples. It also has a flexible structure whose precise operation and configuration are specified by a microinstruction word which arrives at the ALU synchronously with the input data, i.e., under the control of the two-wire interface.

In addition to the 36-bit instruction and 12-bit data input ports, the ALU has a 6-bit run port, and an 8-bit constant port (which actually resides on the token bus). All of these, with the exception of the microinstruction word, drive buses of their respective widths through the There is a single bit within ALU datapath. microinstruction word which represents an extension bit and with the 17-bit-run-sign-level output together There is a two-wire interface at each end of (out data). the ALU datapath, and a set of condition codes which are output together with their own valid signal, cc valid. There is a register file which is accessible to other Huffman Decoder sub-blocks via the ALU, and also to the microprocessor interface.

B.3.2.2 Basic Structure

The basic structure of the Huffman ALU is as shown in Figure 126. It comprises the following components:

Input block 400
Output block 401
Condition Codes block 402
"A" register 403 with source multiplexing
Run register (6 bits) 404 with source multiplexing
Adder/Subtractor 405 with source multiplexing
Sign Extend logic 406 with source multiplexing
Register file 407

20

Each of these blocks (except the output block) drives its output onto a bus running through the datapath, and these buses are, in turn, used as inputs to the multiplexing for block sources. For example, the adder output has it own datapath bus which is one of the possible inputs to the A register. Likewise, the A register has its own bus which forms one of the possible inputs to the adder. Only a subset of all possibilities exist in this respect, as specified in Section 7 on the microinstruction word.

In a single cycle, it is possible to execute either an add-based instruction or a sign-extend-based instruction. Furthermore, it is allowable to execute both of these in a single cycle provided that their operation is strictly parallel. In other words, add then sign extend or sign extend then add sequences are not allowed. The register file may be either read from or written to in a single cycle, but not both.

The output data has three fields:

- ·run 6 bits
- ·sign 1 bit
- level 10 bits

If data is to be passed straight through the ALU, the least significant 11 bits of the input data register are latched into the sign and level fields.

It is possible to program limited multi-cycle operations of the ALU. In this regard, the number of cycles required is given by the contents of the register file location whose address is specified in the microinstruction, and the same operation is performed repeatedly while an iteration counter decrements to one. This facility is typically used to effect left shifts, using the adder to add the A register to itself and to store the result back in the A register.

B.3.3 The Adder/Subtractor Sub-Block

This is a 12-bit wide adder, with optional invert on its input2 and optional setting of the carry-in bit. Output is a .12 bit sum, and carry-out is not used. There are 7 modes

of operation:

ADD: add with carry in set to zero: input1 + input2

ADC: add with carry in set to one: input1 + input2+1

·SBC: invert input2, carry in set to zero: input1 -

5 input2 - 1

10

20

30

SUB: invert input2, carry in set to one: input1 - input2

·TCI: if input2<0, use SUB, else use ADD. This is used with input1 set to zero for obtaining a magnitude value from a two's compliment value.

DCD (DC difference): if input2<0 do ADC, otherwise do ADD.

VRA (vector residual add): if input1<0 do ADC, otherwise do SBC.

15 B.3.4 The Sign Extend Sub-Block

This is a 12-bit unit which sign extends, in various modes, the input data from the size input. Size is a 4 bit value ranging from 0 to 11 (0 relates to the least significant bit, 11 to the most significant). Output is a 12 bit modified data value, and the "sign" bit.

In SGXMODE=NORMAL, all bits above (and including) the size-th bit, take the value of the size-th bit. All those below remain unchanged. Sign takes the value of the size-th bit. For example:

25 data = 1010 1010 1010

size = 2

output = 0000 0000 0010, sign=0

In SGXMOD=INVERSE, all bits above (and including) the size-th bit, take the inverse of the size-th bit, while all those below remain unchanged. Sign takes the inverse of the size-th bit. For example:

data = 1010 1010 1010

size = 0

output = 1111 1111 1111, sign = 1

In SGXMODE=DIFMAG, if the size-th bit is zero, all the bits below (and including) the size-th bit are inverted, while all those above remain unchanged. If the size-th bit

n11 18

25

30

35

is one, all bits remain unchanged. In both cases, sign takes the inverse of the size-th bit. This is used for obtaining the magnitude of AC difference values. For example:

5 data = 0000 1010 1010

size = 2

output = 0000 1010 1101, sign = 1

data = 0000 1010 1010

size = 1

10 output = 0000 1010 1010, sign = 0

In SGXMODE=DIFCOMP, all bits above (but not including) the size-th bit, take the inverse of the size-th bit, while all those below (and including) remain unchanged. Sign takes the inverse of the size-th bit. This is used for obtaining two's compliment values for DC difference values. For example:

data = 1010 1010 1010

size = 0

output = 1111 1111 1110, sign = 1

20 B.3.5 Condition Codes

There are two bytes (16 bits) of condition codes used by the Huffman block, certain bits of which are generated by the ALU/register file. These are the Sign condition code, the Zero condition code, the Extension condition code and a Change Detect bit. The last two of these codes are not really condition codes since they are not used by the Parser in the same way as the others.

The Sign, Zero and Extension condition codes are updated when the Parser issues an instruction to do so, and for each of these instructions the condition code valid signal is pulsed high once.

The Sign condition code is simply the sign extend sign output latched, while the Zero condition code is set to 1 if the input to the A register is zero. The Extension condition code is the input extension bit latched regardless of OUTSRC.

35

Condition codes may be used to evaluate certain condition types:

- result equals constant use subtract and Zero condition
- 5 result equals register value use subtract and Zero condition
 - register equals constant use subtract and Zero
 condition
 - · register bit set use sign extend and Sign condition
- result bit set use sign extend and Sign condition

Note that when using the sign extend and Sign condition code combination, it is possible only to evaluate a single specified bit, rather than multiple bits as would be the case with a conventional logical AND.

- The Change Detect bit, in the present invention, is generated using the same logic as for the Zero condition code, but it does not have an associated valid signal. A bit in the microinstruction indicates that the Change Detect bit should be updated if the value currently being written to the register file is different from that already present (meaning that two clock cycles are necessary, first with REG-MODE set to READ and second with REGMODE set to WRITE). A microprocessor interrupt can then be initiated if a changed value is detected. The Change Detect bit is
- reset by activating Change Detect in the normal way, but with REGMODE set to READ.

The hardwired macroblock counter structure (which forms part of the register file- see below) also generates condition codes as follows: Mb_Start, Pattern_Code, Restart and Pic Start.

B.3.6 The Register File

The address map for the register file is shown below. It uses a 7-bit address space, which is common to both the ALU datapath and the UPI. A number of locations are not accessed by the ALU, these typically being counters in the hardwired macroblock structure, and registers within the ALU itself. The latter have dedicated access, but form

10

15

20

25

30

35

part of the address map for the UPI. Some multi-byte locations (denoted in the table by "O" for oversize) have a single ALU address, but multiple UPI addresses. Similarly, groups of registers which are indexed by the component count, CC (Indicated by I" in the table) are treated as a single location by the ALU. This eases microprogramming for initialization and resetting, and also for block-level operations.

All of the locations, except the dedicated ALU registers (UPI read only), are read/write, and all of the counters are reset to zero by a bit in the instruction word. The pattern code register has a right shift capability, its least significant bit forming the Pattern_Code condition bit. All registers in the hardwired macroblock structure are denoted in the table by "M", and those which are also counters (n-bit) are annotated with Cn.

In the present invention, certain locations have their contents hardwired to other parts of the Huffman subsystem-coding standard, two r-size locations, and a single location (2-bit word) for each of ac huff table and dc huff table to the Huffman Decoder.

Addresses in bold indicate that locations are accessible by both the ALU and the UPI, otherwise they have UPI access only. Groups of registers that are undirected through CC by the ALU can have a single ALU address specified in the instruction word and CC will select which physical location in the group to access. The ALU address may be that of any of the registers in the group, though conventionally, the address of the first should be used. This is also the case for multi-byte locations which should be accessed using the lowest address of the pair, although in practice, either address will suffice. Note that locations 2E and 2F are accessible in the top-level address map (denoted "T"), i.e., not only through the keyhole registers. These two locations are also reset to zero.

The register file is physically partitioned into four "banks" to improve access speed, but this does not affect

the addressing in any way. The main table shows allocations for MPEG, and the two repeated sections give the variations for JPEG and H.261 respectively.



	Adde	Location	T		Ad	dr Location		
						1000000		
	00	A register 1	T	1	3E	c2		
	01	A register 0		1	3F	c3		
	02	run		1,0	40	dc pred_0 1		! -
	10	horiz pels 1		1.0	41.	dc pred_0 0		
	11	horiz pels 0		1,0	42	dc pred_1 :		
	12	vert pels 1	┪	1.0	43	dc pred_1 0		
	13	veπ pels 0	+	1.0	44	dc pred_2 1		!
	14	buff size 1	+	1,0	45	dc pred_2 0		
	15	buff size 0	<u> </u>	1.0	46	dc pred_3 1		
	16	pel asp. ratio	+	1.0	47			
	17	bit rate 2		0	50	dc pred_3 0		
	18	bit rate 1	+	1 0	51	prev mhf 1		
	19	bit rate 0	 	0	52	prev mnf 0		
	1A	pic rate	+ -	0	53	prev mvf 1		<u> </u>
	18	constrained		0		bten unit 0		
	1C	picture type	-		54	prev mhb 1		
	10	H261 picture type	├	10	55	prev mnb 0		
	15	broken closed	 -	0	56	prev mýb 1		
	1F	pred mode	 	0	57	bten wap 0		
	20	vov delay 1		M	60	mp horiz ent1	C13	1
	21	vbv delay 0	 	M	61	Otro Short Gtt		
	22	full pel fwd	 	M	62	mb vert cat 1	C13	1
	23	full pel bwd	 	M	63	mo vert cn:0	•	1
	24	horiz mb copy		M	64	horiz mb 1		1
	25	pic number		M	65	pous up 0		
	26	max h		M	66	vert mb 1		
	27	max v		М	67	vert mb 0		
	28	•		М	68	restart count1	C16	
	29			M	69	restart count0		1
	2A	•		М	6A	restart gap 1		l
	28			M	68	restart gap0		
	2C	first group		M	6C	horiz blk count	C2	
	2D	in picture		M	6D	vert blk count	C2	l T
T.A	2E	rom control		H,M	6E	comp id	C2	
7.R	12F	rom revision	<u> </u>	M	6F	max como id		1
I.H	30	dc huff 0		H.R	70	coding std		
1	31	de nuff 1		M.H	71	pattern code	SRS	
	132	de huff 2		<u>н</u>	72	fwd r size		
<u> </u>	33			Н	73	bwd r size		
I,H	134	do nuff 3 ac nuff 0						!
1	35							1
-	36	ac huff 1 ac huff 2		141	70			1
	1	Table B.3.1 Table		M,I	78	hO		!

Table B.3.1 Table 1: Huffman Register File Address Map

ı	37	ac nuff 3	M,1	79	h1	
1	38	190	M.I	7A	h2	
1	39	101	I.M.I	78	h3	
Ť	I3A	192	M,I	7C	VO.	
<u> </u>	38	193	M,I	70	v1	
<u> </u>	3C	c0	M,I	7E	v2	
<u>,</u>	3D	c1	M.I	7F	v3	

Table B.3.1 Table 1: Huffman Register File Address Map JPEG Variations:

	10	horiz pels l
	11	horiz pels 0
1	12	vert pels 1
	13	vert pels 0
	14	buff size 1
	15	buff size 0
	16	pel asp. ratio
	17	bit rate 2
	18	bit rate 1
	19	bit rate 0
	1A	pic rate
	1B	constrained
	1C	picture type
	1D	H261 picture type
	1E	broken closed
	1F	pred mode
	20	vbv delay l
	21	vbv delay 0
	22	pending frame ch
	23	restart index
	24	horiz mb copy
	25	pic number
	26	max h
	27	max v
	28	·
	29	-
	2A	•

Table B.3.2 JPEG Variations

2B	-
2C	first scan
2D	in picture
2E	rom control
 2F	rom revision

Table B.3.2 JPEG Variations

H.261 Variations

S	10	horiz pels 1	1 —
	11	horiz pels 0	
	12	vert pels 1	
	13	vert pels 0	
	14	buff size 1	
	15	buff size 0	
	16	pel asp. ratio	
	17	bit rate 2	
	18	bit rate 1	
	19	bit rate 0	
	1A	pic rate	
	1B	constrained	
	IC	picture type	
	ID	H261 picture type	
	1E	broken closed	
	lF	pred mode	
	20	vbv delay 1	
	21	vbv delay 0	
	22	full pel fwd	
	23	full pel bwd	
	24	horiz mb copy	
	25	pic number	
	26	max h	
	27	max v	
	28	•	
	29	•	
	2A	•	
	2B	in gob	

10

20	С	first group	
2	D	in picture	
2.	Ε	rom control	
21	F	rom revision	

Table B.3.3 H.261 Variations

B.3.7 The Microinstruction Word

The ALU microinstruction word, in accordance with the present invention, is split into a number of fields, each controlling a different aspect of the structure described above. The total number of bits used in the instruction word is 36, (plus 1 for the extension bit input) and a minimum of encoding across fields has been adopted so that maximum flexibility of hardware configuration is maintained. The instruction word is partitioned as detailed below. The default field values, that is, those which do not alter the state of the ALU or register file, are those given in the *italics*.

Field	Value	Description	Bits	
OUTSRC	RSA6	run, sign. A register as 6 bits	0000	
(specifies	ZZA	zero, zero, A register	0001	
sources for	ZZA8	zero, zero, A register Is 8 bits	0010	
run, sign and	ZZADDU4	zero, zero, adder o/p ms 4 bits	0011	
level output)	ZINPUT	zero, input data	0100	
	RSSGX	run, sign, sign extend o/p	0111	
	RSADD	run, sign, adder o/p	1000	
	RZADD	run, zero, adder o/p	1001	
	RIZADD	input run, zero, adder output		
	ZSADD	zero, sign, adder o/p	1010	
	ZZADD	zero, zero, adder o/p	1011	
	NONE	no valid output - out_valid set to zero	11XX	
REGADDR	00 - 7F	register file address for ALU access	7 bits	
REGSRC	ADD	drive adder o/p onto register file i/p	0	
	SGX	drive sign extend o/p onto register file i/p	l	
REGMODE	READ	read from register file	0	
	WRITE	write to register file	l	
CNGDET	TEST	update change detect if REGMODE is WRITE		
(change	HOLD	do not update change detect bit		
detect)	CLEAR	reset change detect if REGMODE is READ	0	

Table B.3.4 Table 2: Huffman ALU microinstruction fields

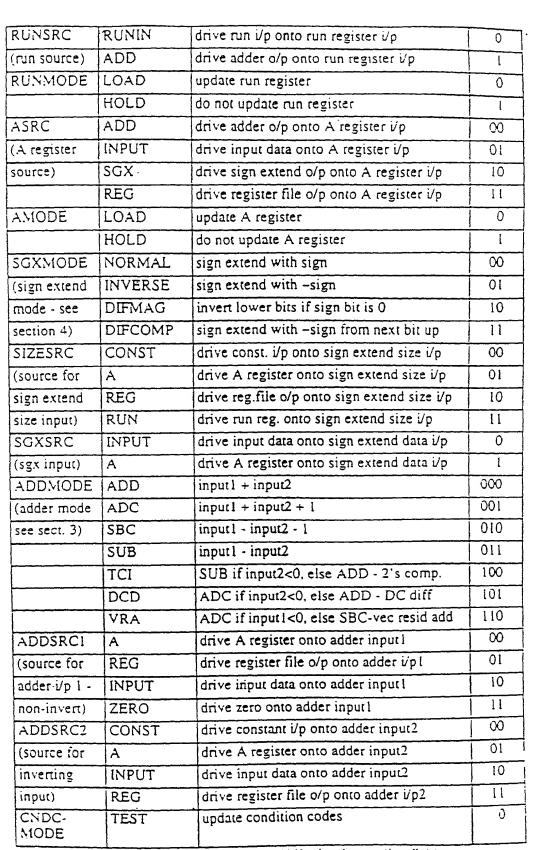


Table 8.3.4 Table 2: Huffman ALU microinstruction fields

(cond. codes)	HOLD	do not update condition codes	1
CNTMODE	NOCOUNT	do not increment counters	X00
(mbstructure	BCINCR	increment block counter and ripple	001
count mode)	CCINCR	force the component count to incr	010
	RESET	reset all counters in mb structure	011
	DISABLE	disable all counters	1XX
INSTMODE	MULTI	iterate current instr multi times	0
	SINGLE	single cycle instruction only	l

Table B.3.4 Table 2: Huffman ALU microinstruction fields

15

20

SECTION B.4 Buffer Manager

B.4.1 Introduction

This document describes the purpose, actions and implementation of the Buffer Manager, in accordance with the present invention (bman).

B.4.2 Overview

The buffer manager provides four addresses for the DRAM interface. These addresses are page addresses in the DRAM. The DRAM interface maintains two FIFOs in the DRAM, the Coded Data Buffer and the Token Data Buffer. Hence, for the four addresses, there is a read and a write address for each buffer.

B.4.3 Interfaces

The Buffer Manager is connected only to the DRAM interface and to the microprocessor. The microprocessor need only be used for setting up the "Initialization registers" shown in Table B.4.4. The interface with the DRAM interface is the four eighteen bit addresses controlled by a REQuest/ACKnowledge protocol for each address. (Since the Buffer Manager is not in the datapath, the Buffer Manager lacks a two-wire interface.)

Furthermore, the Buffer Manager operates off the DRAM interface clock generator and on the DRAM interface scan chain.

25 B.4.4 Address Calculation

The read and write addresses for each buffer are generated from 9 eighteen bit registers:-

Initialization registers (RW from microprocessor)

- *BASECB base address of coded data buffer
- 30 :LENGTHCB maximum size (in pages of coded data buffer
 - ·BASETB base address of token data buffer LENGTHTB maximum size (in pages) of token data buffer
- Dynamic registers (RO from microprocessor)

 READCB coded data buffer read pointer relative to

. .

BASECB

5

35

- NUMBERCB coded data buffer write pointer relative to READCB
- READTB token data buffer read pointer relative to BASETB
- ·NUMBERTB token data buffer write pointer relative
 to READTB

To calculate addresses:-

readaddr = (BASE + READ) mod LIMIT

writeaddr = (((READ + NUMBER) mod LENGTH) + BASE) mod
LIMIT

The "mod LIMIT" term is used because a buffer may wrap around DRAM.

B.4.5 Block Description

15 In the present invention, and as shown in Figure 127, the Buffer Manager is composed of three top level modules connected in a ring which snooper monitors the DRAM interface connection. The modules are beprtize (prioritize), bainstr (instruction), and barecalc (recalculate) are arranged 20 in a ring of that order and omsnoop (snoopers) is arranged on the address outputs. The module, Bapttize, deals with the REQ/ACK protocol, the FULL/EMPTY flags for the buffers and it maintains the state of each address, i.e., "is it a valid address?". From this information, it dictates to 25 bminstr which (if any) address should be recalculated. also operates the BUF CSR (status) microprocessor register, showing FULL/EMPTY flags, and the buf access microprocessor register, controlling microprocessor write access to the buffer manager registers.

The module, Bminstr, on being told by bmprtize to calculate an address, issues six instructions (one every two cycles) to control bmrecalc to calculating an address.

The module, Berecalc, recalculates the addresses under the instruction of beinstr. Running an instruction every two cycles, it contains all of the initialization and dynamic registers, and a simple ALU capable of addition, subtraction and modulus. It informs sheprtize of FULL/EMPTY

states it detects and when it has finished calculating an address.

B.4.6 Block Implementation

B.4.6.1 Bmprtize

At reset, the buf_access microprocessor register is set to one to allow the setting up of the initialization registers. While buf_access reads back one, no address calculations are initiated because they are meaningless without valid initialization registers.

Once buf_access is de-asserted (write zero to it) bmprtize goes about making all the addresses valid (by recalculating them) since its purpose is to keep all four addresses valid. At this stage, the Buffer Manager is "starting up" (i.e., all addresses have not yet been calculated), thus, no requests are asserted. Once all addresses have become valid start-up ends and all requests are asserted. From this point forward, when an address becomes invalid (because it has been used and acknowledged) it will be

No prioritizing between addresses will ever need to be performed, because the DRAM interface can, at its fastest, use an address every seventeen cycles, while the Buffer Manager can recalculate an address every twelve cycles. Therefore, only one address will ever be invalid at one time after start-up. Accordingly, bmprtize will recalculate any invalid address that is not currently being calculated.

In the invention, start-up will be re-entered whenever buf_access is asserted and, therefore, no addresses will be supplied to the DRAM interface during microprocessor accesses.

B.4.6.2 Bminstr

30

35

recalculated.

The module, Bminstr, contains a MOD 12 cycle counter (the number of cycle it takes to generate an address). Note that even cycles start an instruction, whereas odd cycles end an instruction. The top 3 bits along with whether it is a read or a write calculation are decoded into instructions for bmrecalc as follows:

process of the contract of

For read addresses:

Cyc!e	Operation	SusA	Sus8	Result	Meaning of
					result's sign
0-1	ADD	READ	BASE		
2-3	MOD	Accum	LIMIT	Address	
4-5	ADD	READ	1-1-		
6-7	MOD	Accum	LENGTH	READ	
8-9	sua	NUMBER	-1-	NUMBER	
10-11	MOD	-0-	Accum		SET_EMPTY
					(NUMBER >= 0)

Table B.4.1 Read address calculation

For write addresses:

Cycle	Operation	BusA	BusB	Result	Meaning of result's sign
0-1	ADD	NUMBER	READ		
2-3	MOD	Accum	LIMIT		
4-5	ADO	Accum	BASE		
6-7	МОВ	Accum	LIMIT	Address	
8-9	ADD	NUMBER	*1*	NUMBER	
10-11	моо	Accum-	LENGTH		SET_FULL
	1				(NUMBER
					>=
					LENGTH)

Table B.4.2 For write address calculations

 $(x,y) \in \mathbb{R}^{n}$, where $(x,y) \in \mathbb{R}^{n}$, we have $(x,y) \in \mathbb{R}^{n}$, where $(x,y) \in \mathbb{R}^{n}$

15

25:

30

35

Note: The result of the last operation is always held in the accumulator.

When there is no addresses to be recalculated, the cycle counter idles at zero, thus causing an instruction that writes to none of the registers. This has no affect.

B.4.6.3 Bmrecalc

The module, Bmrecalc, performs one operation every two clock cycles: It latches in the instruction from bminstr (and which buffer and io type) on an even counter cycle (start_alu_cyc), and latches the result of the operation on an odd counter cycle (end_alu_cyc). The result of the operation is always stored in the "Accum" register in addition to any registers specified by the instruction. Also, on end_alu_cyc, bmrecalc informs bmprtize as to whether the use of the address just calculated will make the buffer full or empty, and when the address and full/empty has been successfully calculated (load_addr).

Full/empty are calculated using the sign bit of the operation's result.

The modulus operation is not a true modulus, but A mod B is implemented as:

(A>B? (A-B):A)

however this is only wrong when

A > (2B-1)

which will never occur.

B.4.6.4 Bmsnoop

The module, Bmsnoop, is composed of four eighteen bit super snoopers that monitor the addresses supplied to the DRAM interface. The snooper must be "super" (i.e., can be accessed with the clocks running) to allow on chip testing of the external DRAM. These snoopers must work on a REQ/ACK system and are, therefore, different to any other on the device.

REQ/ACK is used on this interface, as opposed to a two-wire protocol because it is essential to transmit information (i.e., acknowledges) back to the sender which an accept will not do). Hence, this rigorously monitors

manager and the second

the FIFO pointers.

B.4.7 Registers

To gain microprocessor write access to the initialization registers, a one should be written to buf_access, and access will be given when buf_access reads back one. Conversely, to give up microprocessor write access, zero should be written to buf_access. Access will be given when buf_access reads back zero. Note that buf_access is reset to one.

The dynamic and initialization registers of the present invention may be read at any time, however, to ensure that the dynamic registers are not changing the microprocessor, write access must be gained.

It is intended that the initialization registers be written to only once. Re-writing them may cause the buffers to operate incorrectly. However, it is envisioned to increase the buffer length on-the-fly and to have the buffer manager use the new length when appropriate.

No check is ever made to see that the values in the initialization registers are sensible, e.g., that the buffers do not overlap. This is the user's responsibility.

15

process as a

r - 1

ji ij i i



Register Name	Usage	Address
CED_BUF_ACCESS	xxxxxO	0x24
CED_BUF_KEYHOLE_ADDR	∞ 000000	0×25
CED_BUF_KEYHOLE	0000000	0×26
CED_BUF_CB_WR_SNP_2	xxxxxxxx	0x54
CED_BUF_CB_WR_SNP_1	5000000	0x55
CED_BUF_CB_WR_SNP_0	30300330	0x56
CED_BUF_CB_RO_SNP_2	xxxxxxx	0x57
CED_BUF_CB_RD_SNP_1	00000000	0x58
CED_BUF_CB_RD_SNP_0	מממממממ	0x59
CED_SUF_TB_WR_SNP_2	xxxxxxDD	0x5a
CED_BUF_TB_WR_SNP_1	מממממממ	0x5b
CED_BUF_TB_WR_SNP_0	DDDCDDDD	0x5c
CED_BUF_TB_RD_SNP_2	xxxxxDD	0×5d
CED_BUF_TB_RO_SNP_1	DDDDDDDDD	0x5e
CED_BUF_TB_RD_SNP_0	מממממממ	0x51

Table B.4.3 Buffer manager non-keyhole registers

Where D indicates a registers bit and \mathbf{x} shows no register bit.

Keyhole Register Name	Usage	Key hole Address
CED_BUF_CB_BASE_3	xxxxxx	0x00
CED_BUF_CB_BASE_2	xxxxxxDD	0x01
CED_BUF_CB_BASE_1		0x02
CED_BUF_CB_BASE_0	00000000	0x03
CED_BUF_CB_LENGTH_3	xxxxxxx	0x04
CED_BUF_CB_LENGTH_2	xxxxxxDD	0x05
CED_BUF_CB_LENGTH_1	מממממממ	0x06
CED_BUF_CB_LENGTH_0	מממכככם	0x07
CED_SUF_CB_READ_3	xxxxxxx	0x08
CED_SUF_CB_READ_2	xxxxxxDD	0x09
CED_BUF_CB_READ_1	00000000	0x0a
CED_SUF_CB_READ_0	0000000	0x0b
CED_BUF_CB_NUMBER_3	xxxxxxx	0x0c

Table B.4.4 Registers in buffer manager keyhole

		T .
Keyhole Register Name	Usage	Key hate Address
CED_BUF_CB_NUMBER_2	xxxxxxx	0x0d
CED_BUF_CB_NUMBER_1	99000000	0x0e
CED_BUF_CB_NUMBER_0	סכסססכככ	10x0
CED_BUF_TB_BASE_3	xxxxxxx	0x10
CED_BUF_TB_BASE_2	XXXXXXXD	0x11
CED_BUF_TB_BASE_1	סכממכניםכ	0x12
CED_BUF_TB_BASE_0	ספכבכבכ	0x13
CED_BUF_TB_LENGTH_3	xxxxxxx	0x14
CED_BUF_TB_LENGTH_2	· xxxxxx00	0x15
CED_BUF_TB_LENGTH_1	0000000	0x16
CED_BUF_TB_LENGTH_0	DDCCCODD	0x17
CED_BUF_TB_READ_3	xxxxxxx	0x18
CED_BUF_TB_READ_2	xxxxxxx	0x19
CED_BUF_TB_READ_1	פסססססס	0x1a
CED_BUF_TB_READ_0	0000000	Oxib
CED_BUF_TB_NUMBER_3	xxxxxxx	0x1c
CED_BUF_TB_NUMBER_2	xxxxxxDD	0x1d
CED_BUF_TB_NUMBER_1	0000000	0x1e
CED_BUF_TB_NUMBER_0	acadacac	0x11
CED_BUF_LIMIT_3	xxxxxxx	0×20
CED_BUF_LIMIT_2	xxxxxxx	0:21
CED_BUF_LIMIT_1	0000000	0x22
CED_BUF_LIMIT_0	סמכמכממ	0×23
CED_BUF_CSR	xxxxx	0x24

Table B.4.4 Registers in buffer manager keyhole

B.4.8 Verification

Verification was conducted in Lsim with small FIFO's onto a dummy DRAM interface, and in C-code as part of the top level chip simulation.

and the second s

B.4.9 Testing

Test coverage to the bman is through the snoopers in bmsnoop, the dynamic registers (shown in B.4.4) and using the scan chain which is part of the DRAM interface scan chain.

SECTION B.5 Inverse Modeler

B.5.1 Introduction

This document describes the purpose, actions and implementation of the Inverse Modeller (imodel) and the Token Formatter (hsppk), in accordance with the present invention.

Note: hsppk is a hierarchically part of the Huffman Decoder, but functionally part of the Inverse Modeller. It is, therefore, better discussed in this section.

10 B.5.2 Overview

15

20

30

The Token buffer, which is between the imodel and hsppk, can contain a great deal of data, all in off-chip DRAM. To ensure that efficient use is made of this memory, the data must be in a 16 bit format. The Formatter "packs" the data from the Huffman Decoder into this format for the Token buffer. Subsequently, the Inverse Modeler "unpacks" data from the Token buffer format.

However, the Inverse Modeller's main function is the expanding out of "run/level" codes into a run of zero data followed by a level. Additionally, the Inverse Modeller ensures that DATA tokens have at least 64 coefficients and it provides a "gate" for stopping streams which have not met their start-up criteria.

B.5.3 Interfaces

25 B.5.3.1 Hsppk

In the present invention, Hsppk has the Huffman Decoder as input and the Token buffer as output. Both interfaces are of the two-wire type, the input being a 17 bit token port, the output being 16 bit "packed data", plus a FLUSH signal. In addition, Hsppk is clocked from the Huffman clock generator and, thus, connected to the Huffman scan chain.

B.5.3.2 Imodel

Imodel has the Token buffer start-up output gate logic (bsogl) as inputs and the Inverse Quantizer as output. Input from the Token buffer is 16 bit "packed data", plus block end signal, from the bsogl is one wirestream enable.

Output is an 11 bit token port. All interfaces are controlled by the two-wire interface protocol. Imodel has its own clock generator and scan chain.

Both blocks have microprocessor access only to the snoopers at their outputs.

B.5.4 Block description

B.5.4.1 Hsppk

Hsppk takes in the 17 bit data from the Huffman and outputs 16 bit data to the Token buffer. This is achieved by first, either truncating or splitting the input data into 12 bit words, and second by packing these words into a 16 bit format.

B.5.4.1.1 Splitting

Hsppk receives 17 bit data from the Inverse Huffman.

This data is formatted into 12 bits using the following formats.

Where F = specifies format; E = extension bit; R = Run bit; L = length bit (in sign mag.) or non-DATA token bit; x = don't care.

20 FLLLLLLLLLFormat 0

ELLLLLLLLLFormat Oa

FRRRRRR00000Format 1

Normal tokens only occupy the bottom 12 bits, having the form:

25 EXXXXXLLLLLLLLL

This is truncated to format Oa

However, DATA tokens have a run and a level in each word in the form:

ERRRRRLLLLLLLLLL.

30 This is broken in to the formats:

ERRRRRLLLLLLLLL->FRRRRRR00000Format 1

ELLLLLLLLFormat Oa

Or if the run is zero format 0 is used: E000000LLLLLLLLLLL->FLLLLLLLLLLLFormat 0

It can be seen that in the format 0, the extension bit is lost and assumed to be one. Therefore, it cannot be used where the extension is zero. In this case, format 1 is

10

unconditionally used.

B.5.4.1.2 Packing

After splitting, all data words are 12 bits wide. Every four 12 bit words are "packed" into three 16 bit words:

Input words	Output words	
0000000000	3003030000001111	
11111111111	111111112222222	
2222222222	222233333333333	
3333333333		

Table B.5.1 Packing method

B.5.4.1.3 Flushing of the buffer

The DRAM interface of the present invention collects a block, 32 sixteen bit "packed" words, before writing them to the buffer. This implies that data can get stuck in the DRAM interface at the end of a stream, if the block is only partially complete. Therefore a flushing mechanism is required. Accordingly, .Hsppk signals the DRAM interface to write it current partially complete block unconditionally.

B.5.4.2.1 Imup (UnPacker)

15 Imup performs three functions:

4) Unpacking data from its sixteen bit format into 12 bit words.

Input words -	Output words
000000000001111	00000000000
111111112222222	11111111111
222233333333333	2222222222
	3333333333

Table B.5.2 Unpacking method

20

30

5) Maintaining correct data during flushing of the Token buffer.

When the DRAM interface flushes, by unconditionally writing the current partially complete block, rubbish data remains in the block. The imup must delete rubbish data, i.e., delete all data from a FLUSH token, until the end of a block.

6) Holding back data until Start-up Criteria are met.

output of data from the block is conditional that a "valid" (stream_enable) is accepted from the Buffer Start-up for each different stream. Consequently, twelve bit data is output to hsppk.

B.5.4.2.2 Imex (EXpander)

In the invention, 'mex expands out all run length codes into runs of zeros followed by a level.

B.5.4.2.3 Impad (PADder)

Impad ensures that all DATA Token bodies contain 64 (or more) words. It does this by padding the last word of the Token with zeros. DATA Tokens are not checked for having over 64 words in the body.

B.5.5 Block implementation

B.5.5.1 Hsppk

Typically, both the Splitting and packing is done in a single cycle.

25 B.5.5.1.1 Splitting

First, the format must be determined IF (datatoken)

IF (lastformat == 1) use format Oa;

ELSE IF (run == 0) use format 0;

ELSE use format 1;

ELSE use format 0a;

and format bit determined

format 0 format bit = 0;

format Oa format bit = extension bit;

35 format 1 format bit = 1;

If format 1 is used, no new data should be accepted in the next cycle because the level of the code has yet to be

output.

B.5.5.1.2 Packing

The packing procedure cycles every four valid data The sixteen bit word output is formed from the last valid word, which is held, and the succeeding word. If this is not valid, then the output is not valid. procedure is:

	Heid Word	Succeeding Word	Packed Word	
valid cycle 0	xxxxxxxxxx	00000000000	XXXXXXXXXXXXX	don't cutcut
valid cycle 1	00000000000	111111111111	0000000000001111	output
valid cycle 2	1111111111111	2222222222	111111112222222	output
valid cycle 3	2222222222	33333333333	2222333333333333	output

Table B.5.3 Packing procedure

Where x indicates undefined bits.

During valid cycle 0, no word is output because it is not 10 valid.

The valid cycle number is maintained by a ring counter. It is incremented by valid data from the splitter and an accepted output.

When a FLUSH (or picture end) token is received and the token itself is ready to output, a flush signal is also output to the DRAM interface to reset the valid cycle to zero. If a FLUSH token arrives on anything but cycle 3, the flush signal must be delayed a valid cycle to ensure the token itself it output.

B.5.5.2 Imodel

15

20

B5.5.2.1 Imup (Unpacker)

As with the packer, the last valid input is stored, and combined with the next input, allows unpacking.

1 11

	Succeeding word	Held Word	Unpacked Word	
valid cycle 0	00000000000001111	XXXXXXXXXXXXXX	000000000000	input
valid cycle 1	111111122222222	0000000000001111	111111111111	ingut
valid cycle 2	22223333333333333	111111112222222	22222222222	don timput
valid cycle 3	2222333333333333	11111111122222222	3333333333333	input

Table B.5.4 Unpacking procedure

Where x indicates undefined bits

The valid cycle is maintained by a ring counter. The unpacked data contains the token's data, flush and PICTURE_END decoded from it. Additionally, format and extension bit are decoded from the unpacked data.

formatbit_is_extn = (lastformat == 1) ll databody
format = databody && (formatbit && lastformatbit)
for token decoding and to be passed on to imex.

When a FLUSH (or picture_end) token is unpacked and output to imex, all data is deleted (Valid forced low)
 until the block end signal is received from the DRAM interface.

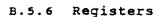
B.5.5.2.2 Imex (EXpander)

- In accordance with the present invention, imex is a four state machine to expand run/level codes out. The state machine is:
 - state0: load run count from run code.
 - state 1: decrement run count, outputting zeros.
- 20 state 2: input data and output levels; default state.
 - state 3: illegal state.

B.5.5.2.3 Impad (PADder)

Impad is informed of DATA Token headers by imex. Next, it counts the number of coefficients in the body of the token.

If the token ends before there are 64 coefficients, zero coefficients are inserted at the end of the token to complete it to 64 coefficients. For example, unextended data headers have 64 zero coefficients inserted after them. DATA tokens with 64 or more coefficients are not affected by impad.



The imodel and happy of the present invention do not have microprocessor registers, with the exception of their snooper.

Register Name	Usage	Address
CED_H_SNP_2	VAxxxx	0x49
CED_H_SNP_1	00000000	0x4a
CED_H_SNP_0	DDDDDDDD	0x4b
CED_IM_SNP_1	VAEXXDCD	0x4a
CED_IM_SNP_0	DDDDDDDD	0x4d

Table B.5.5 Imodel & hsppk registers

Where V = valid bit; A = accept bit; E = extension bit; D = data bit.

B.5.7 Verification

Selected streams run through Lsim simulations.

10 B.5.8 Testing

Test coverage to the imodel at the input is through the Token buffer output snooper, and at the output through the imodel's own snooper. Logic is covered the imodel's own scan chain.

The output of the happy is accessible through the huffman output snooper. The logic is visible through the huffman scan chain.

SECTION B.6 Buffer Start-up

B.6.1 Introduction

This section describes the method and implementation of the buffer start-up in accordance with the present invention.

B.6.2 Overview

5

10

15

To ensure that a stream of pictures can be displayed smoothly and continuously a certain amount of data must be gathered before decoding can start. This is called the start-up condition. The coding standard specifies a VBV delay which can be translated, approximately, into the amount of data needed to be gathered. It is the purpose of the "Buffer Start-up" to ensure that every stream fulfills its start-up condition before its data progresses from the token buffer, allowing decoding. It is held in the buffers by a notional gate (the output gate) at the output of the token buffer (i.e., in the Inverse Modeler). This gate will only be open for the stream once its start-up condition has been met.

20 B.6.3 Interfaces

Bscntbit (Buffer Start-up bit counter) is in the datapath, and communicates by two-wire interfaces, and is connected to the microprocessor. It also branches with a two-wire interface to bsogl (Buffer Start-up Output Gate Logic).

25 Bsogl via a two-wire interface controls image (Inverse Modeler UnPacker), which implements the output gate.

B.6.4 Block Structure

As shown in Figure 130, Bscntbit lies in the datapath between the Start Code Detector and the coded data buffer.

- This single cycle block counts the valid words of data leaving the block and compares this number with the start-up condition (or target) which will be loaded from the microprocessor. When the target is met, bsogl is informed. Data is unaffected by bscntbit.
- 35 Bsogl lies between bscntbit and image (in the inverse modeler). In effect, it is a queue of indicators that streams have met their targets. The queue is moved along

15

20

25

30

35



by streams leaving the buffers (i.e., FLUSH tokens received in the data stream at imup), when another "indicator" is accepted by imup. If the queue is empty (i.e., there are no streams in the buffers which have yet met their start-up target) the stream in imup is stalled.

The queue only has a finite depth, however, this may be indefinitely expanded by breaking the queue in bsogl and allowing the microprocessor to monitor the queue. These queue mechanisms are referred to as internal and external queues respectively.

B.6.5 Block Implementation

B.6.5.1 Bsbitcht (Buffer Start-up bit counter)

Bscntbit counts all the valid words that are input into the buffer start-up. The counter (bsctr) is a programmable counter of 16-24 bits width. Moreover, bsctr has carry look ahead circuitry to give it sufficient speed. Bsctr's width is programmed by ced_bs_prescale. It does this by forcing bits 8-16 high, which makes them always pass a carry. They are, therefore, effectively not used. Only the top eight bits of bsctr are used for comparisons with the target (ced_bs_target).

The comparison (ced_bs_count >=ced_bs_target) is done by bscmp.

The target is derived from the stream when the stream is in the Huffman Decoder and calculated by the microprocessor. It will, therefore, only be set sometime after the start of the stream. Before start-up, the target_valid is set low. Writing to ced_bs_target sets target_valid high and allows comparisons in bscmp to take place. When the comparison shows ced_bs_count >= ced_bs_target, target_valid is set low. The target has been met.

When the target is met the count is reset. Note, it is not reset at the end of a stream. In addition, counting is disabled after the target is met if it is before the end of the stream. The count saturates to 255.

When a stream ends (i.e., a flush) is detected in

30

35



ends before the target is met, an additional event is also generated (bs_flush_before_target_met_event). When any of these events occur, the block is stalled. This allows the user to recommence the search for the next stream's target or in the case of a bs_flush_before_target_met_event event either:

- 1) write a target of zero which will force a target_met or
- 2) note that target was not met and allow the next stream to proceed until this combined with the last stream reaches the target. The target for this next stream can should adjusted accordingly.

B.6.5.2 BSOGL (buffer start-up output gate logic)

As previously described, Bsogl is a queue of indicators that a stream has met its target. The queue type is set by ced_bs_queue (internal(0) or external(1)). This is a reset to select an internal queue. The depth of the queue determines the maximum number of satisfied streams that can be in the coded data buffer, Huffman, and token buffer. When this number is reached (i.e. the queue is full) bsogl will force the datapath to stall at bsbitcht.

Using an internal queue requires no action from the microprocessor. However, if it is necessary to increase the depth of the queue, an external queue can be set (by setting ced_bs_access to gain access to ced_bs_queue which should be set, target_met_event and stream_end_event enabled and access relinquished).

external queue (a count maintained the microprocessor) is inserted into the internal queue. The maintained queue is. by two target met event and stream_end_event. These can simply be referred to as service_queue_input and service queue output respectively] and a register ced bs enable nxt stream. effect, target met_event is the up stream end of the supplying the queue. internal queue Similarly, ced bs enable nxt_stream is the down stream end of the internal queue consuming the queue. Similarly, stream_end_event is a request to supply the down stream queue; stream_end_event resets ced_bs_enable_nxt_stream. The two events should be serviced as follows:

```
/* TARGET_MET_EVENT */
  j = micro_read(CED_BS_ENABLE_NXT_STM);
  if (j == 0) /*Is next stream enabled ?*/
  (/*no, enable it*/
   micro_write(CED_BS_ENABLE_NXT_STM, 1);
   printf(" enable next stream (queue = 0x%x)\n". (context->queue: .
  }
  else /*yes. increment the queue of "target_met" streams*/
   queue++;
   printf(" stream already enabled (queue = 0x3x)\n", (context-
  >queuel);
  }
  /* STREAM_EVENT */
  if (queue > 0) /*are there any "target_mets" left? "/
  (/*yes, decrement the queue and enable another stream */
   queue--;
   micro_write(CED_BS_ENABLE_NXT_STM, 1);
   printf(' enable next stream (queue = 0x%x)\n', (context->queue.
   }
  else
 printf(" queue empty cannot enable next stream (queue = 0x3x)\n".
queue);
micro_write(CED_EVENT_1, 1 << BS_STREAM_END_EVENT); /* clear event
+/
```

The queue type can be changed from internal to external at any time (by the means described above), but they can only be changed external to internal when the external queue is empty (from above "queue==0"), by setting ced_bs_access to gain access to ced_bs_queue which should be reset, target_met_event and stream_end_event masked, and access relinquished.

On the other hand, disable checking of stream start-up conditions, set ced_bs_queue (external), mask target_met_event and stream_end_event and set ced_bs_enable_nxt_stream. In this way, all streams will always be enabled.

B.6.6 Microprocessor registers

Register name	Usage	Address
CED_BS_ACCESS	xxxxxxx	0x10
CED_BS_PRESCALE	xxxxxx	0x11
CED_8S_TARGET	ממממממממ	0x12
CED_BS_COUNT'	סכסססססס	0x13
BS_FLUSH_EVENT	:::::Dr:	0x02
BS_FLUSH_MASK	rrrrDrr	0x03
BS_FLUSH_BEFORE_TARGET_ME	rrrorr	0x02
T_EVENT		
BS_FLUSH_BEFORE_TARGET_ME	::::D:::	0x03
T_MASK		

Table B.6.1 Bscntbit registers

Register name	Usage	Address	
TARGET_MET_EVENT	TTTDTTTT	0x02	
TARGET_MET_MASK	rrrorrr	0x03	
STREAM_END_EVENT	FEDEREE	0x02	
STREAM_END_MASK	EEDEEEE	0x03	

Table B.6.2 Bsogl registers

Register name	Usage	Address	
CED_8S_QUEUE.	xxxxxxx	0x14	
CED_BS_ENABLE_NXT_STM*	xxxxxxx0	0x15	

TableB.6.2 Bsoglegisters

where

- ·D is a register bit
- ·x is a non-existent register bit
- r is a reserved register bit
- to gain access to these registers ced_bs_access must be set to one and polled until it reads back one, unless in an interrupt service routine. Access is given up by setting ced_bs_access to zero.

a the contract

SECTION B.7 The DRAM Interface

B.7.1 Overview

10

15

20

25

30

35

In the present invention, the Spatial Decoder, Temporal Decoder and Video Formatter each contain a DRAM interface block for that particular chip. In all three devices, the function of the DRAM interface is to transfer data from the chip to the external DRAM and from the external DRAM into the chip via block addresses supplied by an address generator.

The DRAM interface typically operates from a clock which is asynchronous to both the address generator and to the clocks of the various blocks through which data is passed. This asynchronism is readily managed, however, because the clocks are operating at approximately the same frequency.

Data is usually transferred between the DRAM Interface and the rest of the chip in blocks of 64 bytes (the only exception being prediction data in the Temporal Decoder). Transfers take place by means of a device known as a "swing buffer". This is essentially a pair of RAMs operated in a double-buffered configuration, with the DRAM interface filling or emptying one RAM while another part of the chip empties or fills the other RAM. A separate bus which carries an address from an address generator is associated with each swing buffer.

Each of the chips has four swing buffers, but the function of these swing buffers is different in each case. In the Spatial Decoder, one swing buffer is used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, one swing buffer is used to write Intra or Predicted picture data to the DRAM, the second to read Intra or Predicted data from the DRAM and the other two to read forward and backward prediction data. Video Formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data from the DRAM, one for each of Luminance (Y) and the Red and

15

20

25

30

35

0.00

Blue color difference data (Cr and Cb, respectively).

The following section describes the operation of a DRAM interface in accordance with the present invention, which has one write swing buffer and one read swing buffer, which is essentially the same as the operation of the Spatial Decoder DRAM Interface. This is illustrated in Figure 131, "DRAM Interface,".

B.7.2 A Generic DRAM Interface

Referring to Figure 131, the interfaces to the address generator 420 and to the blocks which supply and take the data are all two wire interfaces. The address generator 420 may either generate addresses as the result of receiving control tokens, or it may merely generate a fixed sequence of addresses. The DRAM interface 421 treats the two wire interfaces associated with the address generator in a special way. Instead of keeping the accept line high when it is ready to receive an address, it waits for the address generator to supply a valid address, processes that address and then sets the accept line high for one clock implements a request/acknowledge period. Thus, it (REQ/ACK) protocol.

A unique feature of the DRAM Interface is its ability to communicate with the address generator and the blocks which provide or accept the data completely independent of the other. For example, the address generator may generate an address associated with the data in the write swing buffer, but no action will be taken until the write swing buffer signals that there is a block of data which is ready to be written to the external DRAM 422. However, no action is taken until an address is supplied on the appropriate bus from the address generator. Further, once one of the RAMs in the write swing buffer has been filled with data, the other may be completely filled and "swung" to the DRAM Interface side before the data input is stalled (the two-wire interface accept signal set low).

In understanding the operation of the DRAM Interface of the present invention, it is important to note that in a

10

15

20

25

30

35

properly configured system the DRAM Interface will be able to transfer data between the swing buffers and the external DRAM at least as fast as the sum of all the average data rates between the swing buffers and the rest of the chip.

Each DRAM Interface contains a method of determining which swing buffer it will service next. In general, this will be either a "round robin", in which the swing buffer which is serviced is the next available swing buffer which has less recently had a turn, or a priority encoder in which some swing buffers have a higher priority than others. In both cases, an additional request will come from a refresh request generator which has a higher priority than all the other requests. The refresh request is generated from a refresh counter which can be programmed via the microprocessor interface.

B.7.2.1 The Swing Buffers

Figure 132 illustrates a write swing buffer. The operation is as follows:

- 1) Valid data is presented at the input 430 (data in). As each piece of data is accepted it is written into RAM1 and the address is incremented.
- 2) When RAM1 is full, the input side gives up control and sends a signal to the read side to indicate that RAM1 is now ready to be read. This signal passes between two asynchronous clock regimes, and so passes through three synchronizing flip-flops.
- 3) The next item of data to arrive on the input side is written into RAM2, which is still empty.
- 4) When the round robin or priority encoder indicates that it is the turn of this swing buffer to be read, the DRAM Interface reads the contents of RAM1 and writes them to the external DRAM. A signal is then sent back across the asynchronous interface, as in (2), to indicate that RAM1 is now ready to be filled again.
- 5) If the DRAM Interface empties RAM1 and "swings" it before the input side has filled RAM2, then data can

be accepted by the swing buffer continually, otherwise when RAM2 is filled the swing buffer will set its accept signal low until RAM1 has been "swung" back for use by the input side.

5 6) This process is repeated ad infinitum.

The operation of a read swing buffer is similar, but with input and output data busses reversed.

B.7.2.2 Addressing of External DRAM and Swing Buffers

The DRAM Interface is designed to maximize the available memory bandwidth. Consequently, it is arranged so that each 8x8 block of data is stored in the same DRAM page. In this way full use can be made of DRAM fast page access modes, where one row address is supplied followed by many column addresses. In addition, a facility is provided to allow the data bus to the external DRAM to be 8, 16 or 32 bits wide, so that the amount of DRAM used can be matched to the size and bandwidth requirements of the particular application.

In this example (which is exactly how the DRAM Interface on the Spatial Decoder works), the address generator 20 provides the DRAM Interface with block addresses for each of the read and write swing buffers. This address is used as the row address for the DRAM. The six bits of column address are supplied by the DRAM Interface itself, and these bits are also used as the address for the swing 25 buffer RAM. The data bus to the swing buffers is 32 bits wide, so if the bus width to the external DRAM is less than 32 bits, two or four external DRAM accesses must be made before the next word is read from a write swing buffer or 30 the next word is written to a read swing buffer (read and write refer to the direction of transfer relative to the external DRAM).

The situation is more complex in the cases of the Temporal Decoder and the Video Formatter. These are covered separately below.

B.7.3 DRAM Interface Timing

35

In the present invention, the DRAM Interface Timing block

1.0

15

uses timing chains to place the edges of the DRAM signals to a precision of a quarter of the system clock period. Two quadrature clocks from the phase locked loop are used. These are combined to form a notional 2x clock. Any one chain is then made from two shift registers in parallel, on opposite phases of the "2x clock".

First of all, there is one chain for the page start cycle and another for the read/write/refresh cycles. The length of each cycle is programmable via the microprocessor interface, after which the page start chain has a fixed length, and the cycle chain's length changes as appropriate during a page start.

On reset, the chains are cleared and a pulse is created. This pulse travels along the chains, being directed by the state information from the DRAM Interface. The DRAM Interface clock is generated by this pulse. Each DRAM Interface clock period corresponds to one cycle of the DRAM. Thus, as the DRAM cycles have different lengths, the DRAM Interface clock is not at a constant rate.

Further, timing chains combine the pulse from the above chains with the information from DRAM Interface to generate the output strobes and enables (notcas, notras, notwe, notoe).

. .

t ji

SECTION B.8 Inverse Quantizer

B.8.1 Introduction

This document describes the purpose, actions and implementation of the inverse quantizer, (iq) in accordance with the present invention.

B.8.2 Overview

The inverse quantizer reconstructs coefficients from quantized coefficients, quantization weights and step sizes, all of which are transmitted within the datastream.

10 B.8.3 Interfaces

The 1q lies between the inverse modeler and the inverse DCT in the datapath and is connected to a microprocessor. Datapath connections are via two-wire interfaces. Input data is 10 bits wide, output is 11 bits wide.

15 B.8.4 Mathematics of Inverse Quantization

B.8.4.1 H261 Equations

For blocks coded in intra mode:

$$C_{i} = 8Q_{i} \qquad i = 0$$

$$C_{i} = iq_quant_scale\{2Q_{i} + sign\{Q_{i}\}\}\}$$

$$C_{i} = C_{i} - sign\{C_{i}\} \qquad C_{i} = even$$

$$C_{i} = C_{i} \qquad C_{i} = odd$$

$$C_{i} = min(max(C_{i}, -2048).2047)$$

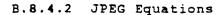
For all other coded blocks:

$$C_{i} = \text{iq_quant_scale}\{2Q_{i} + sign(Q_{i})\}$$

$$C_{i} = C_{i} - sign(C_{i}) \qquad C_{i} = \text{even}$$

$$C_{i} = C_{i} \qquad C_{i} = \text{odd}$$

$$C_{i} = min(max(C_{i} - 2048).2047)$$



$$C_i = W_{i,j}Q_i + 1024$$
 $i = 0$
 $C_i = W_{i,j}Q_i$ $0 < i < 54$
 $C_i = min(max(C_i, -2048).2047)$
 $j = peg_table_indirection(c)$

B.8.4.3 MPEG Equations

For blocks coded in intra mode:

$$C_{i} = W_{i,j}Q_{i} + 1024 \qquad i = 0$$

$$C_{i} = floor\left(\frac{2iq_quant_scaleW_{i,j}Q_{i}}{16}\right)$$

$$C_{i} = C_{i} - sign\left(C_{i}\right) \qquad C_{i} = even \qquad j = 0.2$$

$$C_{i} = C_{i} \qquad C_{i} = odd$$

$$C_{i} = min(max(C_{i}, -2048).2047)$$

1024 is added in intra DC case to account for predictors in huffman being reset to zero.

For all other coded blocks:

$$C_{i} = floor\left(\frac{iq_quant_scaleW_{i,j}\{2Q_{i} + sign(Q_{i})\}}{16}\right)$$

$$C_{i} = C_{i} - sign(C_{i}) \qquad C_{i} = even \qquad \qquad 0 < i < 64$$

$$C_{i} = C_{i} \qquad C_{i} = odd$$

$$C_{i} = min(max(C_{i}, -2048), 2047)$$

B.8.4.4 JPEG Variation Equations

$$C_{i} = floor\left(\frac{2iq_quant_scaleW_{i,j}Q_{i}}{16}\right) + 1024 \qquad i = 0$$

$$C_{i} = floor\left(\frac{2iq_quant_scaleW_{i,j}Q_{i}}{16}\right) \qquad 0 < i < 64$$

$$C_{i} = min(max(C_{i}.-2048).2047)$$

$$j = jpeg_table_indirection(c)$$

B.8.4.5 All other tokens

All tokens except DATA Tokens must pass through the iq unquantized Where:

$$sign(a) = \begin{cases} -1 & a < 0 \\ 0 & a = 0 \\ 1 & a > 0 \end{cases}$$

$$max(a,b) = \begin{cases} a & a > b \\ b & a \le b \end{cases}$$

$$min(a,b) = \begin{cases} a & a \le b \\ b & a > b \end{cases}$$

Floor(a) returns an integer such that:

$$(a-1) < floor(a) \le a$$
 $a \ge 0$
 $a \le floor(a) < (a+1)$ $a \le 0$

Oi are the quantized coefficients.

C_I are the reconstructed coefficients

W_{i,j} are the values in the quantisation table matrices

i is the coefficient index along the zig-zag

j is the quantisation table matrix number (0 \leq j \leq 3)

B.8.4.5 Multiple Standards combined

It can be shown that all the above standards and their variations (also control data which must be unchanged by the iq) can be mapped on to single equation:

OUTPUT =
$$\frac{(2INPUT + k)(xy)}{16}$$

With the additional post inverse quantisation functions of :

- -Add 1024
- •Convert from sign magnitude to 2's complement representation.
- •Round all even numbers to the nearest odd number towards zero.
- *Saturate result to +2047 or -2048.

The variables k, x and y for each variation of the standards and which functions they use is shown in Table B.8.1.



B.8.4.6 Multiple Standards combined

St	andard	x	У		Add	Round	Sat	Convert
		Weight Scale 1024		Even	Pesit	2's camp		
H261	intra DC	8	8	0	No	No	Yes	Yes
	intra	16	iq_quant_scale	1	No	Yes	Yes	Yes
	other	16	iq_quant_scale	1	No	Yes	Yes	Yes
JPEG	DC	W _i	8	0	Yes	No	Yes	Yes
	other	w,	8	0	No	No	Yes	Yes
MPEG	intraOC	8	8	0	Yes	No	Yes	Yes
	intra	W _i	iq_quant_scale	0	No	No	Yes	Yes
	other	w;	iq_quant_scale	1	No	Yes	Yes	Yes
XXX	DC	Wj	iq_quant_scale	0	Yes	No	Yes	
other	W.i	iq_quant_scale	0	No	No I	Yes	Yes	
Other Tok	rens	1	8	0	No	No I	No I	Yes

Table B.8.1 Control decoding

B.8.5 Block Structure

From B.8.4.6 and Table B.8.1, it can be seen that a single architecture can be used for a multi-standard inverse quantizer. Its arithmetic block diagram is shown in Fig. 133 "Arithmetic Block":

Control for the arithmetic block can be functionally broken into two sections:

- Decoding of tokens to load status registers or quantization tables.
 - Decoding of the status registers into control signals.

Tokens are decoded in iqua which controls the next cycle, i.e., iqcb's bank of registers. It also controls the access to the four quantization tables in iquam. The arithmetic, that is, two multipliers and the post functions, are in iquarith. The complete block diagram for the iq is shown in

Figure 134.

B.S.6 Block Implementation

B.8.6.1 Iqca

5

In the invention, iqua is a state machine used to decode tokens into control signals for igram and the register in iqub. The state machine is better described as a state machine for each token since it is reset by each new token. For example:

The code for the QUANT_SCALE (see B.8.7.4, "QUANT_SCALE")

10 and QUANT_TABLE (see B.8.7.6, "QUANT_TABLE") are as
follows:

```
if (tokenheader == QUANT_SCALE)
 sprintf(preport, 'QUANT_SCALE');
 reg_addr = ADDR_IQ_QUANT_SCALE;
 motw = WRITE:
 enable = 1;
}
if (tokenheader == QUANT_TABLE) /*QUANT_TABLE token */
switch (substate)
 case 0: /* quantisation table header */
   sprintf(preport, 'QUANT_TABLE_%s_s0',
    (headerextn ? '(full)' : '(empty)'));
   nextsubstate = 1;
   insertnext = (headerextn ? 0 : 1);
   reg_addr = ADDR_IQ_COMPONENT;
   motw = WRITE:
   enable = 1;
```

```
break;
     case 1: /* quantisation table body */
       sprintf(preport, 'QUANT_TABLE_%s_sl',
        (headerextn ? '(full)' : '(empty)');
      nextsubstate = 1;
      insertnext = (headerextn ? 0 : (qtm_addr_63 == 0));
     reg_addr = USE_QTM;
     rnotw = (headerextn ? WRITE : READ);
     enable = 1;
     break;
   default:
    sprintf(preport, *ERROR in iq quantisation table tokendecoder
(substate %x)\n*,
     substate);
     break;
 }
}
```



Where a substate is a state within a token, QUANT_SCALE has, for example, only one substate. However, the QUANT_TABLE has two, one being the header, the second the token body.

5 The state machine is implemented as a PLA. Unrecognized tokens cause no wordline to rise and the PLA to output default (harmless) controls.

Additionally, iqua supplies addresses to igram from BodyWord counter and inserts words into the stream, for example in an unextended QUANT_TABLE (see B.8.7.4). This is achieved by stalling the input while maintaining the output valid. The words can be filled with the correct data in succeeding blocks (iqub or iqurith).

iqca is a single cycle in the datapath controlled by twowire interfaces.

B.8.6.2 iqcb

10

15

25

30

In the invention, iqcb holds the iq status registers. Under the control of iqca it loads or unloads these from/to the datapath.

The status registers are decoded (see Table B.8.1) into control wires for iqarith; to control the XY multiplier terms and the post quantization functions.

The sign bit of the datapath is separated here and sent to the post quantization functions. Also, zero valued words on the datapath are detected here. The arithmetic is then ignored and zero muxed onto the datapath. This is the easiest way to comply with the "zero in; zero out" spec of the ig.

The status registers are accessible from the microprocessor only when the register iq_access has been set to one and reads back one. In this situation, iqcb has halted the datapath, thus ensuring the registers have a stable value and no data is corrupted in the datapath.

Iqcb is a single cycle in the datapath controlled by two wire interfaces.

B.8.6.3 Igram

· Igram must hold up to four quantization table matrices

5

10

20

30

35

(QTM), each 64*8 bits. It is, therefore, a 256*8 bits six transistor RAM, capable of one read or one write per cycle. The RAM is enclosed by two-wire interface logic receiving its control and write data from igca. It reads out data to igarith. Similarly, igram occupies the same cycle in the datapath as igcb.

The RAM may be read and written from the microprocessor when iq_access reads back one. The RAM is placed behind a keyhole register, iq_qtm_keyhole and addressed by iq_qtm_keyhole_addr. Accessing iq_qtm_keyhole will cause the address to which it points, held in iq_qtm_keyhole_addr to be incremented. Likewise, iq_qtm_keyhole_addr can be written to directly.

B.8.6.4 iqarith

Note, iqurith is three functions pipelined and split over three cycles. The functions are discussed below (see Figure 133).

B.8.6.4.1 XY multiplier

This is a 5(X) by 8(Y) bit carry save unsigned multiplier feeding on to the datapath multiplier. The multiplier and multiplicand are selected with control wires from iqcb. The multiplication is in the first cycle, the resolving adder in the second.

At the input to the multiplier, data from igram can be muxed onto the datapath to read a QUANT_TABLE out onto the datapath.

B.8.6.4.2 (XY) * datapath multiplier

This 13 (XY) by 12 (datapath) bit carry save unsigned multiplier is split over the three cycles of the block. Three partial products in the first cycle, seven in the second and the remaining two in the third.

Since all output from the multiplier is less than 2047 (non_coefficient) or saturated to +2047/-2048, the top twelve bits don't ever need to be resolved. Accordingly, the resolving adder is just two bits wide. On the remainder of the high order bits, a zero detect suffices as a saturate signal.



B.8.6.4.3 Post quantization functions

The post quantization functions are

· Add 1024

5

- Convert from sign magnitude to 2's complement representation.
- Round all even numbers to the nearest odd number towards zero.
- ·Saturate result to +2047 or -2048.
- Set output to zero (see B.8.6.2)
- The first three functions are implemented on a 12 bit adder (pipelined over the second and third cycles). From this, it can be seen what each function requires and these are then combined onto the single adder.

Function	if datapath > 0	il datapath > 0	
Convert to 2's complement	nothing	invert and one	
Round all even numbers	subtract one	add one	

Function	- if datapath > 0	if datacath > 0
Add 1024	add 1024	add 1024

Table B.8.2 Post quantization adder functions

As will be appreciated by one of ordinary skill in the art, care should be taken when reprogramming these functions as they are very interdependent when combined.

The saturate values, zero and zero+1024 are muxed onto the datapath at the end of the third cycle.

20 B.8.7 Inverse Quantizer Tokens

The following notes define the behavior of the Inverse Quantizer for each Token tp which it responds. In all cases, the Tokens are also transported to the output of the

A RESERVE AND A STREET AND ASSESSMENT OF REAL PROPERTY.

Inverse Quantizer. In most cases, the Token is unmodified by the Inverse Quantizer with the exceptions as noted below. All unrecognized Tokens are passed unmodified to the output of the Inverse Quantizer.

5 B.8.7.1 SEQUENCE START

This Token causes the registers iq_prediction mode[1:0] and iq_mpeg_indirection[1:0] to be reset to zero.

B.8.7.2 CODING STANDARD

This Token causes iq_standard[1:0] to be loaded with the appropriate value based upon the current standard (MPEG, JPEG or H.261) being decoded.

B.8.7.3 PREDICTION MODE

This Token loads iq-prediction_mode[1:0]. Although the PREDICTION_MODE Token carries more than two bits, the Inverse Quantizer only needs access to the two lowest order bits. These determine whether or not the block is intra coded.

B.8.7.4 QUANT SCALE

This Token loads iq quant scale[4:0].

20 B.8.7.5 DATA

30

In the present invention, this Token carries the actual quantized coefficients. The head of the token contains two bits identifying the color component and these are loaded into iq_component[1:0]. The next sixty four Token words contain the quantized coefficients. These are modified as a result of the inverse quantization process and are replaced by the reconstructed coefficients.

If exactly sixty four extension words are not present in the Token, the behavior of the Inverse Quantizer is undefined.

The DATA Token at the input of the Inverse Quantizer carries quantized coefficients. These are represented in eleven bits in a sign-magnitude format (ten bits plus a sign bit). The value "minus zero" should not be used but is correctly interpreted as zero.

The DATA Token at the output of the Inverse Quantizer carries reconstructed coefficients. These are represented

1

10

15

20

in twelve bits in a twos complement format (eleven bits plus a sign bit). The DATA Token at the output will have the same number of Token Extension words as it had at the input of the Inverse Quantizer.

5 B.8.7.6 QUANT TABLE

This Token may be used to load a new quantization table or to read out an existing table. Typically, in the Inverse Quantizer, the Token will be used to load a new table which has been decoded from the bit stream. The action of reading out an existing table is useful in the forward quantizer of an encoder if that table is to be encoded into the bit stream.

The Token Head contains two bits identifying the table number that is to be used. These are placed in iq_component[1:0]. Note that this register now contains a "table number" not a color component.

If the extension bit of the Token Head is one, the Inverse Quantizer expects there to be exactly sixty four extension Token Words. Each one is interpreted as a quantization table value and placed in a successive location of the appropriate table, starting at location zero. The ninth bit of each extension Token word is ignored. The Token is also passed to the output of the Inverse Quantizer, unmodified, in the normal way.

If the extension bit of the Token Head is zero, then the Inverse Quantizer will read out successive locations of the appropriate table starting at location zero. Each location becomes an extension Token word (the ninth bit will be zero). At the end of this operation, the Token will contain exactly sixty four extension Token words.

The operation of the Inverse Quantizer in response to this token is undefined for all numbers of extension words except zero and sixty four.

B.8.7.7 JPEG_TABLE_SELECT

This token is used to load or unload translations of color components to table numbers to/from

20

25



507

iq_ipeg_indirection. These translations are used in JPEG
and other standards.

The Token Head contains two bits identifying the color component that is currently of interest. These are placed in iq_component[1:0].

If the extension bit of the Token Head is one, the Token should contain one extension word, the lowest two bits of w h i c h a r e w r i t t e n i n t o t h e iq_ipeg_indirection(2*iq_component[1:0]+1:2*iq_component [1:0]] location. The value just read becomes a Token extension word (the upper seven bits will be zero). At the end of this operation, the Token will contain exactly one Token extension word.

Calour component in nescer	bits of iq_ipeg_indirection accessed
0	[1 0]
1	[3.2]
2	[5:4]
3	[7.5]

Table B.8.3 JPEG TABLE SELECT action

15 B.8.7.8 MPEG TABLE SELECT

This Token is used to define whether to use the default or user defined quantization tables while processing via the MPEG standard. The Token Head contains two bits. Bit zero of the header determines which bit if iq_mpeg_indirection is written into. Bit one is written into that location.

Since the iq_mpeg_indirection[1:0] register is cleared by the SEQUENCE_START Token, it will only be necessary to use this Token if a user defined quantization table has been transmitted in the bit stream.

B.8.8 Microprocessor Registers

B.8.8.1 iq access

To gain microprocessor access to any of the iq registers, iq_access must be set to one and polled until it reads back one (see B.8.6.2). Failure to do this will result in the registers being read still being controlled by the datapath and, therefore, not being stable. In the case of the igram, the accesses are locked out, reading back zeros.

Writing zero to iq_access relinquishes control back to the datapath.

B.8.8.2 Iq_coding_standard[1:0]

This register holds the coding standard that is being implemented by the Inverse Quantizer.

iq_coding_standard	Coding Standard
0	H.261
1	JPEG
2	MPEG
3	xxx

Table B.8.4 Coding standard values

This register is loaded by the CODING_STANDARD Token.

Although this is a two bit register, at present eight bits are allocated in the memory map and future implementations can deal with more than the above standards.

10

25

B.8.8.3 Iq_mpeg_indirection[1:0]

This two bit register is used during MPEG decoding operations to maintain a record of which quantization tables are to be used.

Iq_mpeg_indirection(0) controls the table that is used for intra coded blocks. If it is zero then quantization table 0 is used and is expected to contain the default quantization table. If it is one, then quantization table 2 is used and is expected to contain the user defined quantization table for intra coded blocks.

This register is loaded by the MPEG_TABLE_SELECT Token and is reset to zero by the SEQUENCE_START Token.

B.8.8.4 Iq_ipeg_indirection[7:0]

This eight bit register determines which of the four quantization tables will be used for each of the four possible color components that occur in a JPEG scan.

- *Bits [1:0] hold the table number that will be used for component zero.
- •Bits (3.2) hold the table number that will be used for component one.
- •Bits (5.4) hold the table number that will be used for component two.
- •Bits [7.6] hold the table number that will be used for component three.

This register is affected by the JPEG_TABLE_SELECT Token.

B.8.5 iq_quant_scale[4.0]

This register holds the current value of the quantization scale factor. This register is loaded by the QUANT_SCALE Token.

B.8.8.6 iq component[1:0]

This register usually holds a value which is translated into the Quantization Table Matrix (QTM) number. It is loaded by a number of Tokens.

The DATA Token header causes this register be loaded with the color component of the block which is about to be processed. This information is only used in JPEG and JPEG variations to determine the QTM number, which it does with reference to iq_ipeg_indirection[7:0]. In other standards, iq_component[1:0] is ignored.

The JPEG_TABLE_SELECT Token causes this register be loaded with a color component. It is then used as an index into iq_ipeg_indirection[7:0] which is accessed by the tokens body.

The QUANT_SCALE Token causes this register to be loaded with the QTM number. This table is then either loaded from the Token (if the extended form of the Token is used) or read out from the table to form a properly extended Token.

B.8.8.7 iq prediction mode[1:0]

This two bit register holds the prediction mode that will be used for subsequent blocks. The only use that the Inverse Quantizer makes of this information is to decide whether or not intra coding is being used. If both bits of the register are zero, then subsequent blocks are intra coded.

This register is loaded by the PREDICTION_MODE Token. This register is reset to zero by the SEQUENCE START Token.

Iq_prediction_mode[1:0] has no effect on the operation in JPEG and JPEG variation modes.

20 B.8.8.8 Iq ipeq indirection[7:0]

Iq_ipeg_indirection is used as a lookup table to translate color components into the QTM number. Accordingly, iq_component is used as an index to iq ipeg indirection as shown in Table B.8.3.

This register location is written to directly by the JPEG_TABLE_SELECT Token if the extended form of the Token is used.

This register location is read directly by the JPEG_TABLE_SELECT Token if the non-extended form of the Token is used.

B.8.8.9 Iq_quant_table[3:0][63:0][7:0]

There are four quantization tables, each with 64 locations. Each location is an eight bit value. The value zero should not be used in any location.

These registers are implemented as a RAM described in B.8.6.3, "Igram".

These tables may be loaded using the QUANT_TABLE

Token.

Note that data in these tables are stored in zig-zag scan order. Many documents represent quantization table values as a square eight by eight array of numbers. Usually, the DC term is at the top left with increasing horizontal frequency running left to right and increasing vertical frequency running top to bottom. Such tables must be read along the zig-zag scan path as the numbers are placed into the quantization table with consecutive "i".

10 B.8.9 Microprocessor Register Map

Register	Location	Direction	Reset State
iq_access	0x30	P/W	0
iq_coding_standard(1:0]	0x31	RW	0
iq_quant_scale[4:0]	0x32	RW	?
ig_component(1:0)	0x33	RVW	?
ig_prediction_mode[1:0]	0x34	PVW	0
ic_jpeg_indirection[7:0]	0x35	P/W	?
ra_mpeg_indirection(1:0)	0x36	RW	0
iq_qtm_keynole_addr[7:0]	0x38	P/W	0
rq_qtm_keyhole(7:0)	0x39	RW	?

Table B.8.5 Memory Map

B.8.10 Test

Test coverage to the Inverse Quantizer at the input is through the Inverse Modeler's output snooper, and at the output through the Inverse Quantizer's own snooper. Logic is covered by the Inverse Quantizer's own scan chain.

Access can be gained to igram without reference to iq_access if the ramtest signal is asserted.

SECTION B.9 IDCT

B.9.1 Introduction

The purpose of this description of the Inverse Discrete Cosine Transform (IDCT) block is to provide a source of engineering information for the IDCT. It includes information on the following.

- purpose and main features of the IDCT
- ·how it was designed and verified
- ·structure

5

- It is intended that the description should provide one of ordinary skill in the art sufficient information to facilitate or aid the following tasks.
 - appreciation of the IDCT as a "sillicon macro function"
- 15 · integration the IDCT onto another device
 - development of test programs for the IDCT silicon
 - · modification, re-design or maintenance of the IDCT
 - development of a forward DCT block

B.9.2 Overview

20 A Discrete Cosine Transform/Zig-Zag (DCT/ZZ) performs a transformation on blocks of pixels wherein each block represents an area of the screen 8 pixels high by 8 pixels The purpose of the transform is to represent the pixel block in a frequence domain, sorted according to frequency. Since the eye is sensitive to DC components in 25 a picture, but much less sensitive to high frequency components, the frequency data allows each component to be reduced in magnitude separately, according to the eye's sensitivity. The process of magnitude reduction is known 30 The quantization process reduces the as quantization. information contained in the picture, that quantization process is lossy. Lossy processes give overall data compression by eliminating some information. The frequency data is sorted so that high frequencies, most 35 likely to be quantized to zero, all appear consecutively. The consecutive zeros means that coding the quantized data

by using run-length coding schemes yields further data

15

30

compression, although run-length coding is generally not a lossy process.

The IDCT block (which actually includes an Inverse Zig-Zag RAM, or IZZ, and an IDCT) takes frequency data, which is sorted, and transforms it into spatial data. This inverse sorting process is the function of IZZ.

The picture decompression system, of which the IDCT block forms a part, specifies the pixels as integers. This means that the IDCT block must take, and yield, integer values. However, since the IDCT function is not integer based, the internal number representation uses fractional parts to maintain internal accuracy. Full floating-point arithmetic is preferable, but the implementation described herein uses fixed-point arithmetic. There is some loss of accuracy using fixed-point arithmetic, but the accuracy of this implementation exceeds the accuracy specified by H.261 and the IEEE.

B.9.3 Design Objectives

The main design objective, in accordance with the present invention, was to design a functionally correct IDCT block which uses a minimum silicon area. The design was also required to run with a clock speed of 30MHz under the specified operating conditions, but it was considered that the design should also be adaptable for the future. Higher clock rates will be needed in the future, and the architecture of the design allows for this wherever possible.

B.9.4 IDCT Interfaces Description

The IDCT block has the following interfaces.

- ·a 12-bit wide Token data input port
- -a 9-bit wide Token data output port
- ·a microprocessor interface port
- ·a system services input port
- ·a test interface
- 35 resynchronizing signals

Both the Token data ports are the standard Two-Wire Interface type previously described. The widths

. . .

1 1 10

un men i i i

10

15

20

25

30

35

illustrated, refer to the number of bits in the data representation, not the total number of wires in a port. In addition, associated with the input Token data port are the clock and reset signals used for resynchronization to the output of the previous block. There are also two resynchronizing clocks associated with the output Token data port and used by the subsequent block.

The microprocessor interface is standard and uses four bits of address. There are also three externally decoded select inputs which are used to select the address spaces for events, internal registers and test registers. This mechanism provides the flexibility to map the IDCT address space into different positions in different chips. There is also a single event output, idctevent, and two i/o signals, n_derrd and n_serrd, which are the event tristate data wires to be connected externally to the IDCT and to the appropriate bits of the microprocessor notdata bus.

The system services port consists of the standard clock and reset input signals, as well as, the 2-phase override clocks and associated clock override mode select input.

The test interface consists of the JTAG clock and reset signals, the scan-path data and control signals and the ramtest and chiptest inputs.

In normal operation, the microprocessor port is inactive since the IDCT does not require any microprocessor access to achieve its specified function. Similarly, the test interface is only active when testing or verification is required.

B.9.5 The Mathematical Basis for the Discrete Cosine Transformation

In video bandwidth compression, the input data represents a square area of the picture. The transform applied must, therefore, be two-dimensional. Two-dimensional transforms are difficult to compute efficiently, but the two-dimensional DCT has the property of being separable. Separable transforms can be computed along each dimension independent of the other dimensions. This implementation

The state of the s

uses a one-dimensional IDCT algorithm designed specifically for mapping onto hardware; the algorithm is not appropriate for software models. The one-dimensional algorithm is applied successively to obtain a two-dimensional result.

The mathematical definition of the two-dimensional DCT for an N by N block of pixels is as follows:

EQ 10. forward DCT

$$Y(j,k) = \frac{2}{N}c(j)c(k)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}X(m,n)\cos\left[\frac{(2m+1)j\pi}{2N}\right]\cos\left[\frac{(2n+1)k\pi}{2N}\right]$$

EQ 11. inverse DCT

$$X(m,n) = \frac{2}{N} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} c(j) c(k) Y(j,k) \cos \left[\frac{(2m+1)j\pi}{2N} \right] \cos \left[\frac{(2n+1)k\pi}{2N} \right]$$

Where

5

$$j, k = 0, 1, ..., N-1$$

$$c(j) c(k) = \begin{cases} \frac{1}{\sqrt{2}} & j, k = 0\\ 1 & otherwise \end{cases}$$

10

15

20

25

30

The above definition is mathematically equivalent to multiplying two N by N matrices, twice in succession, with a matrix transposition between the multiplications. A one-dimensional DCT is mathematically equivalent to multiplying two N by N matrices. Mathematically the two-dimensional case is:

$$Y = \begin{bmatrix} x \ c \end{bmatrix}^T C$$

Where C is the matrix of cosine terms.

Thus the DCT is sometimes described in terms of matrix manipulation. Matrix descriptions can be convenient for mathematical reductions of the transform, but it must be stressed that this only makes notation easier. Note that the 2/N term governs the DC level. The constants c(j) and c(k) are known as the normalization factors.

B.9.6 The IDCT Transform Algorithm

As subsequently explained in further detail, the algorithm used to compute the actual IDCT transform should be a "fast" algorithm. The algorithm used is optimized for an efficient hardware architecture and implementation. The main features of the algorithm are the use of $\sqrt{2}$ scaling in order to remove one multiplication, and a transformation of the algorithm designed to yield a greater symmetry between the upper and lower sections. This symmetry results in an efficient re-use of many of the most costly arithmetic elements.

In the diagram illustrating the algorithm (Figure 136), the symmetry between the upper and lower halves is evident in the middle section. The final column of adders and subtractors also has a symmetry, the adders and subtractors can be combined with relatively little cost (4 adder/subtractors being significantly smaller than 4 adders + 4 subtractors as illustrated).

Note that all the outputs of a single dimensional transform are scaled by $\sqrt{2}$. This means that the final 2-

10

30

35



dimensional answer will be scaled by 2. This can then be easily corrected in the final saturation and rounding stage by shifting.

The algorithm shown was coded in double precision floating-point C and the results of this compared with a IDCT reference (using straightforward multiplication). A further stage was then used to code a bit-accurate integer version of the algorithm in C (no timing information was included) which could be used to verify the performance and accuracy of the algorithm as it would be implemented on silicon. The allowable inaccuracies of the transform are specified in the H.261 standard and this method was used to exercise the bitaccurate model and measure the delivered accuracy.

15 Figure 137 shows the overall IDCT Architecture in a way that illustrates the commonality between the upper and lower sections and which also shows the points at which intermediate results need to be stored. The circuit is time multiplexed to allow the upper and lower sections to 20 be calculated separately.

B.9.7 The IDCT Transform Architecture

As described previously, the IDCT algorithm is optimized for an efficient architecture. The key features of the resulting architecture are as follows:

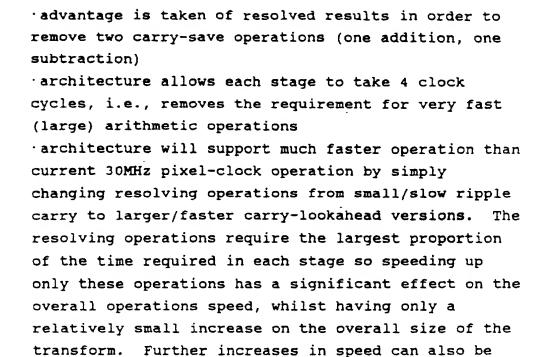
- 25 significant re-use of the costly arithmetic operations
 - small number of multipliers, all being constant coefficient rather than general purpose (reduces multiplier size and removes need for separate coefficient store)
 - small number of latches, no more than required for pipelining the architecture
 - operations are arranged so that only a single resolving operation is required per pipeline stage
 - can arrange to generate results in natural order
 no complex crossbar switching or significant
 multiplexing (both costly in a final implementation)

10

15

30

35



The diagram of the 1D Transform Micro-Architecture (Figure 141) illustrates how the algorithm is mapped onto a small set of hardware resources and then pipelined to allow the necessary performance constraints to be met. The control of this architecture is achieved by matching a "control shift-register" to the data-flow pipeline. This control is straightforward to design and is efficient in silicon layout.

achieved by increasing the depth of pipelining.

control of the transform data-flow is very

straightforward and efficient

The named control signals on Figure 141 (latch,sel_byp etc.) are the various enable signals used to control the latches and, thus, the signal flow. The clock signals to the latches are not shown.

Several implementation details are significant in terms of allowing the transform architecture to meet the required accuracy standards whilst minimizing the transform size. The techniques used generally fall into two major classes.

Retention of maximum dynamic range, with a fixed word width, at each intermediate state by individual

10

15

20

25

30

35

control of the fixed-point position.

·Making use of statistical definition of the accuracy requirement in order to achieve accuracy by selective manipulation of arithmetic operations (rather than increasing accuracy by simply increasing the word width of the entire transform)

The straightforward way to design a transform would involve a simple fixed-point implementation with a fixed word-width made large enough to achieve accuracy. Unfortunately, this approach results in much larger word widths and, therefore, a larger transform. The approach used in the present invention allows the fixed point position to vary throughout the transform in a manner that makes the maximum use of the available dynamic range for any particular intermediate value, achieving the maximum possible accuracy.

Because the allowable results are specified statistically, selective adjustments can be made to any intermediate value truncation operation in order to improve The adjustments chosen are simple overall accuracy. manipulations of LSB calculations, which have little or no cost. The alternative to this technique is to increase the word width, involving significant cost. The adjustments effectively "weight" final results in a given direction, if it is found that previously, these results tend in the opposite direction. By adjusting the fractional parts of results, we are effectively shifting the overall average of these results.

B.9.8 IDCT Block Diagram Description

The block diagram of the IDCT shows all the blocks that are relevant to the processing of the Token Stream. This diagram, Figure 138, does not show details of clocking, test and microprocessor access and the event mechanism. Snooper blocks, used to provide test access, are not shown in the diagram.

B.9.8.1 DATA Error Checker

The first block is the DATA error checker and corrector,

15

20

25

30

35

called "decheck" which takes and produces a 12-bit wide Token Stream, parses this stream and checks the DATA All other Tokens are ignored and are passed straight through. The checks that are performed are for DATA Tokens with a number of extensions not equal to 64. The possible errors are termed "deficient" (<64 extensions) idct too few event, and "supernumerary" extensions), an idct too many event. Such errors are signalled with the standard event mechanism, but the block also attempts simple error recovery by manipulation of the In the case of deficient errors, the DATA Token Stream. Token is packed with "0" value extensions (stops accepting input and performs insert) to make up the correct 64 extensions. In the case of a supernumerary error, the extension bit is forced to "0" for the 64th extension and all extra extensions are removed from the Token Stream.

B.9.8.2 Inverse Zig-Zag

The next block on the Spatial Decoder in Fig. 138 is the inverse zig-zag RAM 441, "izz", and again it takes and produces a 12-bit wide Token Stream. As with all other blocks, the stream is parsed, but only DATA Tokens are recognized. All other Tokens are passed through unchanged. DATA Tokens are also passed through, but the order of the extensions is changed. This block relies on correct DATA Tokens (i.e., 64 extensions only). If this is not true, The reordering is done then operation is unspecified. according to the standard inverse Zig-Zag pattern and, by default, is done so as to provide horizontally scanned data It is also possible to change the at the IDCT output. ordering to provide vertically scanned output. In addition to the standard IZZ ordering, this block performs an extra re-ordering of each 8-word row. This is done because of the specific requirements of the IDCT one-dimensional transform block and results in rows being output in the order (1,3,5,7,0,2,4,6) rather than (0,1,2,3,4,5,6,7).

15

20

30



The next block in Figure 138 is the input formatter 442, "ip_fmt", which formats DATA input for the first dimension of the IDCT transform. This block has a 12-bit wide Token Stream input and 22-bit wide token Stream output. DATA Tokens are shifted left so as to move the integer part to the correct significance in the IDCT transform standard 22-bit wide word, the fractional part being set to 0. This means that there are 10 bits of fraction at this point. All other Tokens are unshifted and the extra unused bits are simply set to 0.

B.9.8.4 1-Dimensional Transform - 1st Dimension

The next block shown in Figure 138 is the first single dimension IDCT transform block 443, "oned". This inputs and outputs 22-bit wide token Streams and, as usual, the stream is parsed and DATA Tokens are recognized. All other tokens are passed through unaltered. The DATA Tokens pass through a pipelined datapath that performs an implementation of a single dimension of an 8-by-8 Inverse Discrete Cosine Transform. At the output of the first dimension, there are 7 bits of fraction in the data word. All other Tokens run through a merely shift register datapath that simply matches the DATA transform latency and are recombined into the Token Stream before output.

25 B.9.8.5 Transpose RAM

The transpose RAM 444 "tram", is similar in many ways to the inverse zig-zag RAM 441in the way it handles a Token Stream. The width of Tokens handled (22 bits) and the reordering performed are different, but otherwise they work in the same way and actually share much of their control logic. Again, rows are additionally re-ordered for the requirements of the following IDCT dimension as well as the fundamental swapping of columns into rows.

B.9.8.6 1-Dimensional Transform - 2nd Dimension

35 The next block shown is another instance of a single dimension IDCT transform and is identical in every way to the first dimension. At the output of this dimension there

20

30

35

are 4 bits of fraction.

B.9.8.7 Round and Saturate

The round-and-saturate block 446 in Figure 138, "ras", takes a 22-bit wide Token Stream containing DATA extensions in 22-bit fixed point format and outputs a 9-bit wide Token Stream where DATA extensions have been rounded (towards +ve infinity) into integers and saturated into 9-bit two's complement representation and all other Tokens have been passed straight through.

B.9.9 Hardware Descriptions of Blocks 10

B.9.9.1 Standard Block Structure

For all the blocks that handle a Token Stream there is a standard notional structure as shown in Figure 139. separates the two-wire interface latches from the section that performs manipulation of the Token Stream. Variations on this structure can include extra internal blocks (such as a RAM core). In some blocks shown, the structure is made less obvious in the schematic (although it does actually still exist) because of the requirement of grouping together all the "datapath" logic and separate this from all the standard cell logic. In the case of a very simple block, such as "ras", it is possible to take the latched out accept straight into the input two-wire latch without logical manipulation.

B.9.9.2 "Decheck" - DATA Error Checking/Recovery 25

The first block 440 in the Token Stream performs DATA checking and correcting as specified in the Block Diagram Overview section. The detected errors are handled with the standard event mechanism which means that events can be masked and the block can either continue with the recovery procedure when an error is detected or be stopped depending on event mask status. The IDCT should never see incorrect DATA Tokens and, therefore, the recovery that it attempted is only a fairly simple attempt to contain what may be a serious problem.

This block has a pipeline depth of two stages and is implemented entirely in zcells. The input two-wire

15

20

25

30

35

interface latch is of the "front" type, meaning that all inputs arrive onto transistor gates to allow safe operation when this block (at the front of the IDCT) is on a separate power supply regime from the one preceding it. This block works by parsing a Token Stream and passing non-DATA Tokens straight through. When a DATA Token is found, a count is started of the number of extensions found after the header. If the extension bit is found to be "0" when the count does not equal 63, an error signal is generated (which goes to the event logic) and depending on the state of the mask bit for that event, "decheck" will either be stopped (i.e., no longer accept input or generate output) or will begin error The recovery mechanism for "deficient" errors uses the counter to control the insertion of the correct number of extensions into the Token Stream (the value inserted is always "0"). Obviously, input is not accepted whilst this insertion proceeds. When it is found that the extension bit is not "0" on the 64th extension. "supernumerary" error is generated, the DATA Token is completed by forcing the extension bit to "0", and all succeeding words with the extension bit set to "1" are deleted from the Token Stream by continuing to accept data but invalidating the output.

Note that the two error signals are not persistent (unless the block is stopped) i.e., the error signal only remains active from the point when an error is detected until recovery is complete. This is a minimum of one complete cycle and can persist forever in the case of a infinitely supernumerary DATA Token.

B.9.9.3 "Izz" and "tram" - Reordering RAMs

The "izz" 441 (inverse zig-zag RAM) and the "tram" 444 (transpose RAM) are considered here together since they both perform a variation on the same function and they have more similarities than differences. Both these blocks take a Token Stream and re-order the extensions of a DATA Token whilst passing through all other Tokens unchanged. The widths of the extensions handled and the sequences of the

15

20

25

30

35

re-ordering are different, but a large section of the control logic for each RAM is identical and is actually organized into a "common control" block which is instanced in the schematic for each RAM. The difference in width has no effect upon this control section so it is only necessary to use a different "sequence address generator" for each RAM together with RAM cores and two-wire interface blocks of the appropriate width.

The overall behavior of each RAM is essentially that of a FIFO. This is strictly true at the Token level and a particular modification to the output order is made for the extension words of a DATA Token. The depth of the FIFO is 128 stages. This is necessary to fulfill the requirement for a sustainable 30 MHz throughout the system since output of the FIFO is held up after the start of the output of a DATA Token is detected. This is because the features of the reordering sequences used require that a complete block of 64 extensions be gathered in the FIFO before re-ordered output can begin. More precisely, the minimum number required is different for inverse ziq-zaq and transpose sequences and is somewhat less than 64 in both cases. However, the complications of controlling a FIFO which has a length which is not a power of two, means that the small saving in RAM core would be outweighed by the additional complexity of control logic required.

The RAM core is implemented with a design which allows a read and a write (to the same or separate addresses) in a single 30 MHz cycle. This means that the RAM is effectively operating with an internal 60 MHz cycle time.

The re-ordering operation is performed by generating a particular sequence of read addresses ("sequence address generation") in the range 0-> 63, but not in natural order. The sequences required are specified by the standard zigzag sequence (for eight horizontal or vertical scanning) or by the sequence needed for normal matrix transposition. These standard sequences are then further reordered by the requirement to output each row in Odd/Even format (i.e.,

15

25

30

1,3,5,7,0,2,4,6) rather than (0,1,2,3,4,5,6,7)) because of the requirements of the IDCT transform 1-dimensional blocks.

Transpose address sequence generation is quite straightforward algorithmically. Straight transpose sequence generation simply requires the generation of row and column addresses separately, both implemented with counters. The row re-ordering requirement simply means that row addresses are generated with a simple specific state machine rather than a natural counter.

Inverse zig-zag sequences are rather less straightforward to generate algorithmically. Because of this fact, a small ROM is used to hold the entire 64 6 bit values of address, this being addressed with row and column counters which can be swapped in order to change between horizontal and vertical scan modes. A ROM based generator is very quick to design and it further has the advantage that it is trivial to implement a forward zig-zag (ROM re-program) or to add other alternative sequences in the future.

20 B.9.9.4 "Oned" - Single Dimension IDCT Transform

This block has a pipeline depth of 20 stages and the pipeline is rigid when stalled. This rigidity greatly simplifies the design and should not unduly affect overall dynamics since the pipeline depth is not that great and both dimensions come after a RAM which provides a certain amount of buffering.

The block follows the standard structure, but has separate paths internally for DATA Token extensions (which are to be processed) and all other items which should be passed through unchanged. Note that the schematic is drawn in a particular way. First, because of the requirements to group together all the datapath logic and second, to allow automatic compiled code generation (this explains the control logic at the top level).

Tokens are parsed as normal and then DATA extensions, and other values, are routed respectively through two different parallel paths before being re-combined with a multiplexer

10

15

20

25

30

35



before the output two-wire interface latch block. The parallel paths are required because it is not possible to pass values unchanged through the transform datapath. The latency of the transform datapath is matched with a simple shift register to handle the remainder of the Token Stream.

The control section of "oned" needs to parse the Token Stream and control the splitting and re-combination of the Tokens. The other major section controls the transform datapath. The main mechanism for the control of this datapath is a control shift-register which matches the datapath pipeline and is tapped-off to provide the necessary control signals for each stage of the datapath pipeline.

The "oned" block has the requirement that it can only start operation on complete rows of DATA extensions, i.e., groups of 8. It is not able to handle invalid data ("Gaps") in the middle of rows, although, in fact, the operation of "izz" and the "tram" ensure that complete DATA blocks are output as an uninterrupted sequence of 64 valid extension values.

B.9.9.4.1 Transform Datapath

The micro-architecture of the transform datapath, "t_dp" was previously shown in Figure 141. Note that some detail (e.g., clocking, shifts, etc.) is not shown. This diagram does illustrate, however, how the datapath operates on four values simultaneously at any stage in the pipeline. The basic sub-Structure of the datapath, i.e., the three main sections can also be seen (e.g., pre-common, common and post-common) as can the arithmetic and latch resources required. The named control signals are the enables for the pipeline latches (and the add/sub selector) which are sequenced with decodes of the control shift-register state. Note that each pipeline stage is actually four clock cycles in length.

within the transform datapath there are a number of latch stages which are required to gather input, store intermediate results in the pipeline, and serialize the

35

output. Some of latches are of the muxing type, i.e., they can be conditionally loaded from more than one source. All the latches are of the enabled type, i.e., there are separate clock and enable inputs. This means that it is easy to generate enable signals with the correct timing, rather than having to consider issues of skew that would arise if a generated clock scheme was adopted.

The main arithmetic elements required are as follows.

- ·a number of fixed coefficient multipliers
- 10 (carry-save output)
 - ·carry-save adders
 - ·carry-save subtractors
 - · resolving adders
 - · resolving adder/subtractors

15 arithmetic is performed in two's complement representation. This can either be in normal (resolved) form or in carry-save form (i.e., two numbers whose sum represents the actual value). All numbers are resolved before storage and only one resolving operation is 20 performed per pipeline stage since this is the most expensive operation in terms of time. The resolving operations performed here all use simple ripple-carry. This means that the resolvers are guite small, relatively slow. Since the resolutions dominate the total 25 time in each stage, there is obviously an opportunity to speed up the entire transform by employing fast resolving arithmetic units.

B.9.9.5 "Ras" - Rounding and Saturation

In the present invention, the "ras" block has the task of taking 22-bit fixed point numbers from the output of the second dimension "oned" and turning these into the correctly rounded and saturated 9-bit signed integer results required. This block also performs the necessary divide-by-4 inherent in the scheme (the 2/N term) and to further divide-by-2 required to compensate for the <2 prescaling performed in each of the two dimensions. This division by 8 implies that the fixed point position is

interpreted as being three bits further left than anticipated, i.e., treat the result as having 15 bits of integer representation and 7 bits of fraction (rather than 4 bits of fraction). The rounding mode implemented is "round to positive infinity", i.e., add one for fractions of exactly 0.5. This is primarily done because it is the simplest rounding mode to implement. After rounding (a conditional increment of the integer part) is complete, this result is inspected to see whether the 9-bit signed result requires saturation to the maximum or minimum value in this range. This is done by inspection of the increment carry out together with the upper bits of the original integer value.

As usual, the Token Stream is parsed and the round and saturation operation is only applied to DATA Token extension values. The block has a pipeline depth of two stages and is implemented entirely in zcells.

B.9.9.6 "Idctsels" - IDCT Register Select Decoder

This block is a simple decoder which decodes the 4 microprocessor interface address lines, and the "sel_test" input, into select lines for individual blocks test access (snoopers and RAMs). The block consists only of zcells combinatorial logic. The selects decoded are shown in Table B.9.2.

10

15

20

Addr.	Bit	
(hex)	num.	Register Name
0x0	71	not used
	0	TRAM keyhole address
0x1	70	The sources
0x2 ·	70	TRAM keyhole data
0x3	70	TRAM keyhole data ^a
0x4	70	IZZ keyhole address
0x5	70	IZZ keyhole data
0x6	73	not used
	2	ipfsnoop test select
	1	ipfsnoop valid
	0	ipfsnoop accept
0x7	75	not used
	50	ipfsnoop bits(21:15)
0x8	70	ipfsnoop bits[15:8]
0x9	70	ipfsnoop bits[7:0]
OxA	73	not used
İ	2	d2snoop test select
	1	d2snoop valid
	0	d2snoop accept
0x8	76	not used
	50	
0xC	70	d2snoop bits[21:16] d2snoop bits[15:8]
0xD	70	
0xE	7	d2snoop bits[7:0]
	6	outsnoop test select
	5	outsnoop valid
}	42	outsnoop accept
0x5	10	not used
0xF	70	outsnoop data(9:8)
		outsnoop data[7:0]

Table B.9.1 IDCT Test Address Space

a. Repeated address

10

15

20

25

30

B.9.9.7 "Idctregs" - IDCT Control Register and Events

This block of the invention contains instances of the standard event logic blocks to handle the DATA deficient and supernumerary errors and also a single memory mapped bit "vscan" which can be used to make the "izz" re-ordering change such that the IDCT output is vertically scanned. This bit is reset to the value "0", i.e., the default mode is horizontally scanned output. The two possible events are OR-ed together to form an idctevent signal which can be used as an interrupt. See Section B.9.10 for the addresses and bit positions of registers and events.

B.9.9.8 Clock Generators

Two "standard" type ("clkgen") clock generators are used in the IDCT. This is done so that there can be two separate scan-paths. The clock generators are called "idctcga" and "idctcgb". Functionally, the only difference is that "idctcgb" does not need to generate the "notrst1" signal. The amounts of buffering for each of the clock and reset outputs in the two clock generators is individually tailored to the actual loads driven by each clock or reset. The loads that are matched were actually measured from the gate and track capacitances of the final layout.

When the IDCT top-level Block Place and Route (BPR) was performed, advantage was taken of the capabilities of the interactive global routing feature to increase the widths of tracks of the first sections of the clock distribution trees for the more heavily loaded clocks (ph0_b and ph1_b) since these tracks will carry significant currents.

B.9.9.9 JTAG Control Blocks

Since the IDCT has two separate scan-chains, and two clock generators, there are two instances of the standard JTAG control block, "jspctle". These interface between the test port and the two scan-paths.

B.9.10 Event and Control Registers

The IDCT can generate two events and has a single bit of control. The two events are idct_too_few_event and idct too many event which can be generated by the "decheck"

block at the front of the IDCT if incorrect DATA Tokens are detected. The single control bit is "vscan" which is set if it is required to operate the IDCT with the output vertically scanned. This bit, therefore, controls the "izz" block. All the event logic and the memory mapped control bit are located in the block "idctregs".

From the point of view of the IDCT, these registers are located in the following locations. The tristate i/o wires n_derrd and n-serrd are used to read and write to these locations as appropriate.

Addr. (hex)	Bit num.	 Register Name
0x0	71	not used
	0	vscan

Table B.9.2 IDCT Control Register Address Space

Addr.	Bit	Register Name
(hex)	name	
0x0	n_derrd	idct_too_few_event
	n_serrd	idct_too_many_event
Cx1	n_derrd	idct_too_few_mask
	n_serrd	idct_too_many_mask

Table B.9.3 IDCT Event Address Space

B.9.11 Implementation Issues

B.9.11.1 Logic Design Approach

In the design of all the IDCT blocks, in accordance with the invention, there was an attempt to use a unified and simple logic design strategy which would mean that it was possible to do a "safe" design in a quick and straightforward manner. For the majority of control logic, a simple scheme of using master-slaves only was adopted. Asynchronous set/reset inputs were only connected to the correct system resets. Although it might often be possible to come up with clever non-standard circuit configurations to perform the same functions more efficiently, this scheme possesses the following advantages.

- · conceptually simple
- 15 easy to design

5

10

speed of operation is fairly obvious (cf.

latch->logic->latch>logic style design) and
amenable to automatic analysis

- glitches not a problem (cf. SR latches)
- 20 using only system reset for initialization
 - allows scan paths to work correctly
 - ·allows automatic complied C-code generation

There are a number of places where transparent d-type latches were used and these are listed below.

25 B.9.11.1.1 two-wire interface latches

The standard block structure uses latches for the input and output two-wire interfaces. No logic exists between an output two-wire latch and the following input two-wire latch.

30 <u>B.9.11.1.2</u> ROM interface

Because of the timing requirements of the ROM circuit, latches are used in the IZZ sequence generator at the output of the ROM.

B.9.11.1.3 Transform Datapath and Control Shift-Register

It is possible to implement every pipeline storage stage as a full master-slave device, but because of the amount of storage required there is a significant savings to be had

by using latches. However, this scheme requires the user to consider several factors.

signals of both phases for use as enables (i.e.,
need to use latches in this shift-register)
timing analysis complicated by use of latches
the "t_postc" will no longer automatically produce
compiled code since one latch outputs to another
latch of the same phase (because of the timing of the
enables this is not a problem for the circuit)
Nonetheless, the area saved by the use of latches makes
it worthwhile to accept these factors in the present
invention.

B.9.11.1.4 Microprocessor interfaces

Due to the nature of this interface, there is a requirement for latches (and resynchronizers) in the Event and register block "idctregs" and in the keyhole logic for RAM cores.

B.9.11.1.5 JTAG Test Control

These standard blocks make use of latches.

B.9.11.2 Circuit Design Issues

Apart from the work done in the design of the library cells that were used in the IDCT design (standard cells, datapath library, RAMs, ROMs, etc.) there is no requirement for any transistor level circuit design in the IDCT. Circuit simulations (using Hspice) were performed of some of the known critical paths in the transform datapath and Hspice was also used to verify the results of the Critical Path Analysis (CPA) tool in the case of paths that were close to the allowed maximum length.

Note that the IDCT is fully static in normal operation (i.e., we can stop the system clocks indefinitely) but there are dynamic nodes in scanable latches which will decay when test clocks are stopped (or very slow). Due to the non-restored nature of some nodes which exhibit a Vt drop (e.g., mux outputs) the IDCT will not be "micro-power" when static.

B.9.11.3 Layout Approach

5

10

15

20

25

30

The overall approach to the layout implementation of the present invention was to use BPR (some manual intervention) to lay out a complete IDCT which consisted of many zcells and a small number of macro blocks. These macro blocks were either hand-edited layout (e.g., RAMs, ROM, clock generators, datapaths) or, in the case of the "oned" block, had been built using BPR from further zcells and datapaths.

Datapaths were constructed from kdplib cells. Additionally, locally defined layout variations of kdplib cells were defined and used where this was perceived as providing a worthwhile size benefit. The datapath used in each of the "oned" blocks, "oned_d", is by far the largest single element in the design and considerable effort was put into optimizing the size (height) of this datapath.

The organization of the transform datapath, "t dp", is rather crucial since the precise ordering of the elements within the datapath will affect the way the interconnect is handled. It is important to minimize the number of "overs" (vertical wires not connecting to a sub-block) which occur at the most congested point since there is a maximum allowed value (ideally 8, 10 is also possible, although highly inconvenient). The datapath is split logically into three major sub-sections and this is the way that the datapath layout was performed. In each subsection, there are really four parallel data flows (which are combined at various points) and there are, therefore, many ways of organizing the flows of data (and, thus, the positions of all the elements) within each subsection. The ordering of the blocks within each subsection, and also the allocation of logical buses to physical bus pitches was worked out carefully before layout commenced in order to make it possible to achieve a layout that could be connected correctly.

35 B.9.12 Verification

The verification of the IDCT was done at a number of levels, from top-level verification of the algorithms to

15

20

25



final layout checks.

The initial work on the transform architecture was done in C, both full-precision and bit-accurate integer models were developed. Various tests were performed on the bit-accurate model in order to prove the conformance to the H.261 accuracy specification and to measure the dynamic ranges of the calculations within the transform architecture.

The design progressed in many cases by writing an M behavioral description of sub-blocks (for example, the control of datapaths and RAMs). Such descriptions were simulated in Lsim before moving onto the design of the schematic description of that block. In some cases (e.g., RAMs, clock generators) the behavioral descriptions were still used for top-level simulations.

The strategy for performing logic simulation was to simulate the schematics for everything that would simulate adequately at that level. The low-level library cells (i.e., zcells and kdplib) were mainly simulated using their behavioral descriptions since this results in far smaller and quicker simulations. Additionally, the behavioral library cells provide timing check features which can highlight some circuit configuration problems. confidence check, some simulations were performed using the transistor descriptions of the library cells. logic simulations were in the zero-delay manner and, therefore, were intended to verify functional performance. The verification of the real timing behavior is done with other techniques.

Lsim switch-level simulations (with RC_Timing mode being used) were done as a partial verification of timing performance, but also provide checks for some other potential transistor level problems (e.g., glitch sensitive circuits).

The main verification technique for checking timing problems was the use of the CPA tool, the "path" option for "datechk". This was used to identify the longer signal

Most Lsim simulations were performed with the standard source->block->sink methodology since the bulk of the IDCT behavior is exercised by the flow of Tokens through the device. Additional simulations are also necessary to test the features accessed through the microprocessor interface (configuration, event and test logic) and those test features accessed via JTAG/scan.

compiled-code simulations can be readily accomplished by one of ordinary skill in the art for entire IDCT, again using the standard source->bloc->sink method and many of the same Token Streams that were used in the Lsim verification.

15 B.9.13 Testing and Test Support

This section deals with the mechanisms which are provided for testing and an analysis of how each of the blocks might be tested.

The three mechanisms provided for test access are as 20 follows:

- ·microprocessor access to RAM cores
- ·microprocessor access to snooper blocks
- scan path access to control and datapath logic

There are two "snooper" blocks and one "super snooper" block in the IDCT. Figure 140 shows the positions of the snooper blocks and the other microprocessor test access.

Using these, and the two RAM blocks, it is possible to isolate each of the major blocks for the purpose of testing their behavior in relation to the Token flow. Using microprocessor access, it is possible to control the Token inputs to any block and then to observe the Token port output of that block in isolation. Furthermore, there are two separate scan paths which run through (almost) all of the flip-flops and latches in the control sections of each block and also some of the datapath latches in the case of the "oned" transform datapath pipeline. The two scan paths are denoted "a" and "b", the former running from the

to a man had been been been been been by the Boll of the second by the Boll of the second by the Boll of the second by the Boll of the second by the Boll of the second by the Boll of the second by t

30

35

5

15

20

25

30

"decheck" block to the "ip_fmt" block and the latter from the first "oned" block to the "ras" block.

Access to snoopers is possible by accessing the appropriate memory mapped locations in the normal manner. The same is true of the RAM cores (using the "ramtest" input as appropriate). The scan paths are accessed through the JTAG port in the normal way.

Each of the blocks is now discussed with reference to the various test issues.

10 B.9.13.1 "Decheck"

This block has the standard structure (see Figure 139) where two latches for the input and output two-wire interfaces surround a processing block. As usual, no scan is provided to the two-wire latches since these simply pass on data whenever enabled and have no depth of logic to be tested. In this block, the "control" section consists of a 1-stage pipeline of zcells which are all on scanpath "a". The logic in the control section is relatively simple, the most complex path is probably in the generation of the DATA extension count where a 6-bit incrementer is used.

B.9.13.2 "Izz"

This block is a variant of the standard structure and includes a RAM core block added to the two-wire interface latches and the control section. The control section is implemented with zcells and a small ROM used for address All the zcells are on scanpath "a" sequence generation. and there is access to the ROM address and data via zcell There is also further logic, e.g., for the generation of numbers plus the ability to increment or decrement. In addition, there is a 7-bit full adder used The RAM core is accessible for read address generation. the microprocessor keyhole registers, via through interface, see Table B.9.1.

B.9.13.3 "lp fmt"

This block again has the standard structure. Control logic is implemented with some rather simple zcell logic (all on scanpath "a") but the latching and shifting/muxing

 of the data is performed in a datapath with no direct access since the logic here is very shallow and simple.

B.9.13.4 "Oned"

Again, this block follows the standard structure and divides into random logic and datapath sections. The zcell logic is relatively straightforward, all the zcells are on scanpath "a". The control signals for the transform pipeline datapath are derived from a long shift register consisting of zcell latches which are on the scanpath. Additionally, some of the pipeline latches are on the scanpath, this being done because there is a considerable depth of logic between some stages of the pipeline (e.g., multipliers and adders). The non-DATA Tokens are passed along a shift register, implemented as a datapath, and there is no test access to any of the stages.

B.9.13.5 Tram'

10

15

20

25.

30

This block is very similar to the "izz" block. In this case, however, there is no ROM used in the address sequence address generation. This is performed algorithmically. All the zcell control states are on datapath "b".

B.9.13.6 Rras'

This block follows the standard structure and is entirely implemented with zcells. The most complex logical function is the 8-bit incrementer used when rounding up. All other logic is fairly simple. All states are scanpath "b".

B.9.13.7 Other top-level blocks

There are several other blocks that appear at the top level of the IDCT. The snoopers are obviously part of the test access logic, as are the JTAG control blocks. There are also the two clock generators which do not have any special test access (although they support various test features). The block "idctsels" is combinatorial zcell logic for decoding microprocessor addresses and the block "idctregs" contains the microprocessor accessible event and control bits associated with the IDCT.

SECTION B.10 Introduction

B.10.1 Overview of the Temporal Decoder

The internal structure of the Temporal Decoder, accordance with the invention, is shown in Figure 142.

All data flow between the blocks of the chip (and much of the data flow within blocks) is controlled by means of the usual two-wire interfaces and each of the arrows in Figure 142 represents a two-wire interface. The incoming token stream passes through the input interface 450 which synchronizes the data from the external system clock to the phase-locked-loop internal clock derived from the (ph0/ph1). The token stream is then split into two paths via a Top Fork 451; one stream passes to the Address Generator 452 and the other to a 256 word FIFO 453. FIFO buffers data while data from previous I or P frames is fetched from the DRAM and processed in the Prediction Filters 454 before being added to the incoming error data from the Spatial Decoder in the Prediction Adder 455 (P and During MPEG decoding, frame reordering data must also be fetched for I and P frames so that the output frames are in the correct order, the reordered data being inserted into the stream in the Read Rudder block 456.

The Address Generator 452 generates separate addresses for forward and backward predictions, reorder, read and write-back, the data which is written back being split from the stream in the Write Rudder block 457. Finally, data is resynchronized to the external clock in the Output Interface Block 458.

All the major blocks in the Temporal Decoder connected to the internal microprocessor interface (UPI) This is derived from the external microprocessor interface (MPI) bus in the Microprocessor Interface block This block has address decodes for the various blocks in the chip associated with it. Also associated with the microprocessor interface is the event logic.

The rest of the logic of the Temporal Decoder is

11 ij 5

10

15

20

25

30

35

concerned primarily with test. First, the IEE 1149.1 (JTAG) interface 460 provides an interface to internal scan paths as well as to JTAG boundary-scan features. Secondly, two-wire interface stages which allow intrusive access to the data flow via the microprocessor interface while in test mode are included at strategic points in the pipeline architecture.

. .

15

20

25

30

35

SECTION B.11 Clocking, Test and Related Issues

B.11.1 Clock Regimes

Before considering the individual functional blocks within the chip, it is helpful to have an appreciation of the clock regimes within the chip and the relationship between them.

During normal operation, most blocks of the chip run synchronously to the signal pllsysclk from the phase-locked-loop (PLL) block. The exception to this is the DRAM interface whose timing is governed by the need to be synchronous to the iftime sub-block, which generates the DRAM control signals (notwe, notoe, notcas, notras). The core of this block is clocked by the two-phase non-overlapping clocks clk0 and clk1, which are derived from the quadrature two-phase clocks supplied independently from the PLL cki0, cki1 and clkq0, ckq1.

Because the clk0, clk1 DRAM interface clocks asynchronous to the clocks in the rest of the chip, measures have been taken to eliminate the possibility of metastable behavior (as far as practically possible) at the interfaces between the DRAM interface and the rest of the The synchronization occurs in two areas: in the chip. output interfaces of the Address Generator (addrgen/predread/psgsync, addrgen/ip wrt2/sync18 addrgervip rd2/sync18) and in the blocks which control the "swinging" of the swing-buffer RAMs in the DRAM Interface (see section on the DRAM Interface). In each case, the synchronization process is achieved by means of three metastable-hard flip-flops in series. It should be noted that this means that clk0/clk1 are used in the output stages of the Address Generator.

In addition to these completely asynchronous clock regimes, there are a number of separate clock generators which generate two-phase non-overlapping clocks (ph0, ph1) from pllsysclk. The Address Generator, Prediction Filters and DRAM Interface each have their own clock generators; the remainder of the chip is run off a common clock

15

20

25

generator. The reasons for this are twofold. First, it reduces the capacitive load on individual clock generators, allowing smaller clock drivers and reduced clock routing widths. Second, each scan path is controlled by a clock generator, so increasing the number of clock generators allows shorter scan-paths to be used.

It is necessary to resynchronize signals which are driven across these clock-regime boundaries because the minor skews between the non-overlapping clocks derived from different clock generators could mean that underlap occurred at the interfaces. Circuitry built into each "Snooper" block (see Section B.11.4) ensures that this does not occur, and Snooper blocks have been placed at the boundaries between all the clock regimes, excepting at the front of the Address Generator, where the resynchronization is performed in the Token Decode block.

B.11.2 Control of Clocks

Each standard clock generator generates a number of different clocks which allow operation in normal mode and scan-test mode. The control of clocks in scan-test mode is described in detail elsewhere, but it is worth noting that several of the clocks generated by a clock generator (tph0, tph1, tckm, tcks) do not usually appear to be joined to any primitive symbols on the schematics. This is because scan paths are generated automatically by a post-processor which correctly connects these clocks. From a functional point of view, the fact that the post-processor has connected different clocks from those shown on the schematics can be ignored; the behavior is the same.

During normal operation, the master clocks can be derived in a number of different ways. Table B.11.1 indicates how various modes can be selected depending on the states of the pins pllselect and override.

pliselect	override	Mode
0	0	pllsyscik is connected directly to external syscik,
		bypassing the PLL; DRAM Interface clocks (cki0, cki1,
		ckq0, ckq1) are controlled directly from the pins ti and tq.
0	1	Override mode - ph0 and ph1 clocks are controlled
		directly from pins tphoish and tph1ish; DRAM interface
		clocks (cki0, cki1, ckq0, ckq1) are controlled directly
		from the pins tl and tq.
1	0	Normal operation, pilsyscik is the clock generated by the
		PLL: DRAM Interface clocks are generated by the PLL.
1	1	External resistors connected to it and tig are used instead
		of the internal resistors (debug only).

Table B.11.1 Clock Control Modes

B.11.3 The Two-wire Interface

The overall functionality of the two-wire interface is described in detail in the Technical Reference. However, the two-wire interface is used for all block-to-block communication within the Temporal Decoder and most blocks consist of a number of pipeline stages, all of which are themselves two-wire interface stages. It is, therefore, essential to understand the internal implementation of the two-wire interface in order to be able to interpret many of the schematics. In general, these internal pipeline stages are structured as shown in Figure 143.

10

15

20

25

30

35

Figure 143 shows a latch-logic-latch representation as this is the configuration which is normally used. However, when a number of stages are put together, it is equally valid to think of a "stage" as being latch-latch-logic (for many engineers a more familiar model). The use of the latch-logic-latch configuration allows all inter-block communication to be latch to latch, without any intervening logic in either the sending or receiving block.

Referring again to Figure 143, a simple two-wire interface FIFO stage can be constructed by removing the logic block, connecting the data and valid signals directly between the latches and the latched in_valid directly into the NOR gate on the input to the in_accept latch in the same way as out_valid and out_accept are gated. Data and valid signals then propagate when the corresponding accept signal is high. By ORing in_valid with out_accept_reg in the manner shown, data will be accepted if in_valid in low, even if out_accept_reg is low. In this way gaps (data with the valid bit low) are removed from the pipeline whenever a stall (accept signal low) occurs.

With the logic block inserted, as shown in Figure 143, in_accept and out_valid may also be dependent on the data or the state of the block. In the configuration shown, it is standard for any state within the block to be held in master-slave devices with the master enabled by phl and the slave enabled by ph0.

B.11.4 Snooper Blocks

Snooper blocks enable access to the data stream at various points in the chip via the Microprocessor Interface. There are two types of snooper blocks. Ordinary Snoopers can only be accessed in test mode where the clocks can be controlled directly. "Super Snoopers" can be accessed while the clocks are running and contain circuitry which synchronizes the asynchronous data from the Microprocessor bus to the internal chip clocks. Table B.11.2 lists the locations and types of all Snoopers in the Temporal Decoder.

Location	Туре
addrgenvec_pipe/snoopz31	Snooper
addrger/cnt_pipe/midsnp	Snooper
addrtgervent_pipe/endsnp	Snooper
addrgen/precread/snoopz44	Snooper
addrgervip_wrt2/superz10	Super Snooper
addrgervip_rc2/superz10	Super Snooper

Table B.11.2 Snoopers in Temporal Decoder.

Location	Туре
dramx/drami/ilsnoops/snoopz15 (Isnp)	Snooper
dramx/dramil/ilsnoops/snoopz15 (bsnp)	Snooper
dramx/dramil/ilsnoops/super29	Super Snooper
wrudder/superz9	Super Snooper
pflts/fwdflVdimbuff/snoopk13	Snooper
pflts/bwdflt.dimbuff/snoopk13	Snooper
pflts/snoopz9	Snooper

Table B.11.2 Snoopers in Temporal Decoder

Details on the use of both Snoopers are contained in the test section. Details of the operation of the JTAG interface are contained in the JTAG document.

SECTION B.12 Functional Blocks

B.12.1 Top Fork

The Top Fork, in accordance with the present invention, serves two different functions. First, it forks the data stream into two separate streams: one to the Address Generator and the other to the FIFO. Second, it provides the means of starting and stopping the chip so that the chip can be configured.

The fork part aspect of the component is very simple. The same data is presented to both the Address Generator and the FIFO, and has to have been accepted by both blocks before an accept is sent back to the previous stage. Thus, the valids of the two branches of the fork are dependent on the accepts from the other branch. If the chip is in a stopped state, the valids to both branches are held low.

The chip powers up in a state where in_accept is held low until the configure bit is set high. This ensures that no data is accepted until the user has configured the chip. If the user needs to configure the chip at any other time, he must set the configure bit and wait until the chip has finished the current stream. The stopping process is as follows:

- 1) If the configure bit has been set, do not accept any more data after a flush token has been detected by the Top Fork.
- 2) The chip will have finished processing the stream when the FLUSH Token reaches the Read Rudder. This causes the signal seq_done to go high.
- 3) When seq_done goes high, set an event bit which can be read by the Microprocessor. The event signal can be masked by the Event block.

B.12.2 Address Generator

In the present invention, the address generator (addrgen)
is responsible for counting the numbers of blocks within a
frame, and for generating the correct sequence of addresses
for DRAM data transfers. The address generator's input is

25

10

15

20



the token stream from the token input port (via topfork), and its output to the DRAM interface consists of addresses and other information, controlled by a request/acknowledge protocol.

- The principal sections of the address generator are:
 - · token decode
 - ·block counting and generation of the DRAM block address
 - ·conversion of motion vector data into an address
- 10 offset
 - request and address generator for prediction transfers
 - reorder read address generator
 - ·write address generator

15 B.12.2.1 Token Decode (tokdec)

In the Token Decoder, tokens associated with coding standards, frame and block information and motion vectors are decoded. The information extracted from the stream is stored in a set of registers which may also be accessed via the upi. The detection of a DATA token header is signalled to subsequent blocks to enable block counting and address generation. Nothing happens when running JPEG.

List of tokens decoded:

- · CODING STANDARD
- 25° · DATA

20

- · DEFINE MAX SAMPLING
- · DEFINE SAMPLING
- · HORIZONTAL MBS
- · MVD BACKWARDS
- 30 · MVD FORWARDS
 - · PICTURE START
 - · PICTURE TYPE
 - · PREDICTION MODE

This block also combines information from the request generators to control the toggling of the frame pointers and to stall the input stream. The stream is stalled when a new frame appears at the input (in the form of a

10

15

20

2 5.

30

35

PICTURE_START token) but the writeback or reorder read associated with the previous frame is incomplete.

B.12.2.2 Macroblock Counter (mblkcntr)

The macroblock counter of the present invention consists of four basic counters which point to the horizontal and vertical position of the macroblock in the frame and to the horizontal and vertical position of the block within the macroblock. At the beginning of time, and on each PICTURE_START, all counters are reset to zero. As DATA Token headers arrive, the counters increment and reset according to the color component number in the token header and the frame structure. This frame structure is described by the sampling registers in the token decoder.

For a given color component, the counting proceeds as follows. The horizontal block count is incremented on each new DATA Token of the same component until it reaches the width of the macroblock, and then it resets. The vertical block count is incremented by this reset until it reaches the height of the macroblock, and then it resets. When this happens, the next color component is expected. Hence, this sequence is repeated for each of the components in the macroblock - the horizontal and vertical size of the macroblock, possibly being different for each component. If, for any component, fewer blocks are received than are expected, the count will still proceed to the next component without error.

When the color component of the DATA Token is less than the expected value, the horizontal <u>macroblock</u> count is incremented. (Note that this will also occur when more than the expected number of blocks appear for a given color component, as the counters will then be expecting a higher component index.) This horizontal count is reset when the count reaches the picture width in macroblocks. This reset increments the vertical macroblock count.

There is a further ability to count macroblocks in H.261 CIF format. In this case, there is an extra level hierarchy between macroblocks and the picture called the

20

25

30

35

group of blocks. This is eleven macroblocks wide and three deep, and a picture is always two groups wide. The token decoder extracts the CIF bit from the PICTURE_TYPE token and passes this to the macroblock counter to instruct it to count groups of blocks. Instances of too few or too many blocks per component will provoke similar reactions as above.

B.12.2.3 Block Calculation (blkcalc)

The Block calculation converts the macroblock and block-within-macroblock coordinates into coordinates for the block's position in the picture, i.e., it knocks out the level of hierarchy. This, of course, has to take into account the sampling ratios of the different color components.

15 B.12.2.4 Base block Address (bsblkadr)

The information from the blkcalc, together with the color component offsets, is used to calculate the block address within the linear DRAM address space. Essentially, for a given color component, the linear block address is the number of blocks down times the width of the picture plus the number of blocks long. This is added to the color component offset to form the base block address.

B.12.2.5 Vector Offset (vec_pipe)

The motion vector information presented by the token decoder is in the form of horizontal and vertical pixel offset coordinates. That is, for each of the forward and backward vectors there is an (x,y) which gives the displacement in half-pixels from the block being formed to the block from which it is being predicted. Note that these coordinates may be positive or negative. They are first scaled according to the sampling of each color component, and used to form the block and new pixel offset coordinates.

In Figure 145, the shaded area represents the block that is being formed. The dotted outline is the block from which it is being predicted. The big arrow shows the block offset - the horizontal and vertical vector to the DRAM

20

25

30

block that contains the prediction block's origin - in this case (1,4). The small arrow shows the new pixel offset - the position of the prediction block origin within that DRAM block. As the DRAM block is 8x8 bytes, the pixel offset looks to be (7,2).

The multiplier array vmarrla then converts the block vector offset into a linear vector offset. The pixel information is passed to the prediction request generator as an (x,y) coordinate (pix_info).

10 B.12.2.6 Prediction Requests

The frame pointer, base block address and vector offset are added to form the address of the block to be fetched from the DRAM (Inblkad3). If the pixel offset is zero, only one request is generated. If there is an offset in either the x OR y dimension, then two requests are generated - the original block address and the one either immediately to the right or immediately below. With an offset in both x and y, four requests are generated.

Synchronization between the chip clock regime and the DRAM interface clock regime takes place between the first addition (Inblkad3) and the state machine that generates the appropriate requests. Thus, the state machine (psgstate) is clocked by the DRAM interface clocks, and its scanned elements form part of the DRAM interface scan chain.

B.12.2.7 Reorder Read Requests and Write Requests

As there is no pixel offset involved here, each address is formed by adding the base block address to the relevant frame pointer. The reorder read uses the same frame store as the prediction and data is written back to the other frame store. Each block includes a short FIFO to store addresses as the transfer of read and write data is likely to lag the prediction transfer at the corresponding address. (This is because the read/write data interacts with stream further along the chip dataflow than the prediction data). Each block also includes synchronization between the chip clock and the DRAM interface clock.

B.12.2.8 Offsets

The DRAM is configured as two frame stores, each of which contains up to three color components. The frame store pointers and the color component offsets within each frame must be programmed via the upi.

B.12.2.9 Snoopers

In the present invention, snoopers are positioned as follows:

- Between blkcalc and bsblkadr this interface comprises
 the horizontal and vertical block coordinates, the
 appropriate color component offset and the width of the
 picture in blocks (for that component).
 - After bsblkadr the base block address.
- After vec_pipe the linear block offset, the
 pixel offset within the block, together with
 information on the prediction mode, color component
 and H.261 operation.
 - ·After Inblkad3 the physical block address, as described under "Prediction Requests".
- Super snoopers are located in the reorder read and write request generators for use during testing of the external DRAM. See the DRAM Interface section for all the details.

 B.12.2.10 Scan
- The addrgen block has its own scan chain, the clocking of which is controlled by the block's own clock generator (adclkgen). Note that the request generators at the back end of the block fall within the DRAM interface clock regime.

B.12.3 **Prediction Filters

The overall structure of the Prediction Filters, in accordance with the present invention, is shown in Figure 146. The forward and backward filters are identical and filter the MPEG forward and backward prediction blocks. Only the forward filter is used in H.261 mode (the h261_on input of the backward filter should be permanently low because H.261 streams do not contain backward predictions). The entire Prediction Filters block is composed of

pipelines of two-wire interface stages.

B.12.3.1 A Prediction Filter

5

10

15

20

25

Each Prediction Filter acts completely independently of the other, processing data as soon as valid data appears at its input. It can be seen from Figure 147 that a Prediction Filter consists of four separate blocks, two of which are identical. It is best if the operation of these blocks is described independently for MPEG and H.261 operation. H.261 being the more complex, is described first.

B.12.3.1.1 H.261 Operation

re pgr

The one-dimensional filter equation used is as follows:

$$F_i = \frac{x_{i+1} + 2x_i + x_{i-1}}{4} (1 \le i \le 6)$$

$$F_i = x_i(otherwise)$$

This is applied to each row of the 8x8 block by the x Prediction Filter and to each column by the y Prediction The mechanism by which this is achieved is Filter. illustrated in Figure 148, which is basically a representation of the pflt1dd schematic. The filter consists of three two-wire interface pipeline stages. For the first and last pixels in a row, registers A and C are reset and the data passes unaltered through registers B, D and F (the contents of B and D being added to zero). control of Bx2mux is set so that the output of register B is shifted left by one. This shifting is in addition to the one place which it is always shifted in any event. Thus, all values are multiplied by 4 (more of this later). For all other pixels, x_{i+1} is loaded into register C, x_i into register B and $\mathbf{x}_{i,1}$ into register A. It can be seen from Figure 148 that the H.261 filter equation is then implemented. Because vertical filtering is performed in

10

15

20

25

horizontal groups of three (see notes on the Dimension Buffer, below) there is no need to treat the first and last pixels in a row differently. The control and the counting of the pixels within a row is performed by the control logic associated with each 1-D filter. It should be noted that the result has not been divided by 4. Division by 16 (shift right by 4) is performed at the input of the Prediction Filters Adder (Section B.12.4.2) after both horizontal and vertical filtering has been performed, so that arithmetic accuracy is not lost. Registers DA, DD and DF pass control information down the pipeline. This includes h261_on and last_byte.

Of the other blocks found in the Prediction Filter, the function of the Formatter is merely to ensure that data is presented to the x-filter in the correct order. It can be seen above that this merely requires a three-stage shift register, the first stage being connected to the input of register C, the second to register B and the third to register A.

Between the x and y filters, the Dimension Buffer buffers data so that groups of three vertical pixels are presented to the y-filter. These groups of three are still processed horizontally, however, so that no transposition occurs within the Prediction Filters. Referring to Figure 149, the sequence in which pixels are output from the Dimension Buffer is illustrated in Table B.12.1.

Clock	Input Pixel	Output Pixel	Clock	Input Pixel	Output Pixel
1	0	55(a)	17	16	7
2	1	56	18	17	
3	2	57	19	18	F (0, 3, 15) (b
4	3	58	20	19	F (1 9, 17)
5	4	59	21	20	F (2, 10 'S)
5	5	60	22	21	F 3 '' '9.
7	5	61	23	22	F 4 '2 20.
8	7	62	24	23	F 5 '3 2'
9	8	63	25	24	F 6, 14 221
10	9	0	26	25	F (7 15, 23)
11	10	1	27	25	F (8, 16 24)
12	11	2	28	27	F 19, 17, 25,
13	12	3	29	28	F (10, 18, 26)
14	13	4	30	29	F-11 19 27
15	14	5	31	30	F -12, 20 28:
16	15	6	32	31	F 12 20 28

Table B.12.1: H.261 Dimension Buffer Sequence

- a. Least row of pixels from previous block or invalid data if there was no previous block (or if there was
 - a long gap between blocks.)
- b. F(x) indicates the function in H.261 filter equation.

5

B.12.3.1.2 MPEG Operation

During MPEG operation, a Prediction Filter performs a simple half pel interpolation:

$$F_i = \frac{x_i + x_{i+1}}{2} (0 \le i \le 8, \text{half pel})$$

$$F_i = x_i (0 \le i \le 7, \text{integer} pel)$$

This is the default filter operation unless the h261_on input is low. If the signal dim into a 1-D filter is low then integer pel interpolation will be performed. Accordingly, if h261_on is low and xdim and ydim are low, all pixels are passed straight through without filtering. It is an obvious requirement that when the dim signal into a 1-D filter is high, the rows (or columns) will be 8 pixels wide (or high). This is summarized in Table B.12.2. Referring to Figure 148, "1-D Prediction Filter,", the

h261_on	xdim	ydim	Function
0	0	0	F, = X,
0	0	1	MPEG 8x9 block
0	1	0	MPEG 9x8 block
0	1	1	MPEG 9x9 block
1	0	0	H.251 Low-pass Filter
1	0	1	lilegal
1	j 1	0	lilegal
1	1	1	Illegal

Table B.12.2 1-D Filter Operation

15

20

25

30

35

operation of the 1-D filter is the same for MPEG inter pel as it is for the first and last pixels in a row in H.261. For MPEG half-pel operation, register A is permanently reset and the output of register C is shifted left by 1 (the output of register B is always shifted left by 1 anyway). Thus, after a couple of clocks register F contains (2B +2C), four times the required result, but this is taken care of at the input of the Prediction Filters Adder, where the number, having passed through both x and y filters, is shifted right by 4.

The function of the Formatter and Dimension Buffer are also simpler in MPEG. The formatter must collect two valid pixels before passing them to the x-filter for half-pel interpolation; the Dimension Buffer only needs to buffer one row. It is worth noting that after data has passed through the x-filter, there can only ever be 8 pixels in a row, because the filtering operation converts 9-pixel rows into 8-pixel rows. "Lost" pixels are replaced by gaps in the data stream. When performing half-pel interpolation, the x-filter inserts a gap at the end of each row (after every 8 pixels); the y-filter inserts 8 gaps at the end of This is significant because the group of 8 or 9 gaps at the end of a block align with DATA Token headers and other tokens between DATA Tokens in the stream coming out of the FIFO. This minimizes the worst-case throughput of the chip which occurs when 9x9 blocks are being filtered.

B.12.3.2 The Prediction Filters Adder.

During MPEG operation, predictions may be formed using an earlier picture, a later picture, or the average of the two. Predictions formed from an earlier frame termed forward predictions and those formed from a later frame are called backward predictions. The function of the Prediction Filters Adder (pfadd) is to determine which filtered prediction values are being used (forward, backward or both) and either pass through the forward or backward filtered predictions or the average of the two

10

15

20

25

30



(rounded towards positive infinity).

The prediction mode can only change between blocks, i.e., at power-up or after the fwd_lst_byte and/or bwd_lst_byte signals are active, indicating the last byte of the current prediction block. If the current block is a forward prediction then only fwd_lst_byte is examined. If it is a backward prediction then only bwd_lst_byte is examined. If it is a bidirectional prediction, then both fwd_lst_byte and bwd_lst_byte are examined.

The signals fwd_on and bwd_on determine which prediction values are used. At any time, either both or neither of these signals may be active. At start-up, or if there is a gap when no valid data is present at the inputs of the block, the block enters a state when neither signal is active.

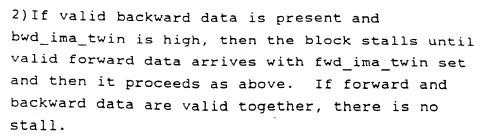
Two criteria are used to determine the prediction mode for the next block: the signals fwd ima twin bwd_ima_twin, which indicate whether a forward or backward block is part of a bidirectional prediction pair, and the buses fwd_p_num[1:0] and bwd_p_num[1:0]. These buses contain numbers which increment by one for each prediction block or pair of prediction blocks. blocks are necessary because, for example, if there are two forward prediction blocks followed by a bidirectional prediction block, the DRAM interface can fetch the backward block of the bidirectional prediction sufficiently far ahead so that it reaches the input of the Prediction Filters Adder before the second of the forward prediction blocks. Similarly, other sequences of backward and forward predictions can get out of sequence at the input of the Prediction Filters Adder. Thus, the next prediction mode is determined as follows:

1) If valid forward data is present and fwd_ima_twin is high, then the block stalls until valid backward data arrives with bwd_ima_twin set and then it goes through the blocks averaging each pair of prediction values.

10

30

35



- 3) If valid forward data is present, but fwd_ima_twin is not set, then fwd_p_num is examined. If this equals the number from the previous prediction plus one (stored in pred_num) then the prediction mode is set to forward.
- 4) If valid backward data is present but bwd_ima_twin is not set, then bwd_p_num is examined. If this equals the number from the previous prediction plus one (stored in pred_num) then the prediction mode is set to backward.

Note that "early_valid" signals from one stage back in the pipeline are used so that the Prediction Filters Adder mode can be set up before the first data from a new block arrives. This ensures that no stalls are introduced into the pipeline.

The ima_twin and pred_num signals are not passed along
the forward and backward prediction filter pipelines with
the filtered data. This is because:

- 1) These signals are only examined when fwd_1st_byte and/or bwd_1st_byte are valid. This saves about 25 three-bit pipeline stages in each prediction filter.
- 2) The signals remain valid throughout a block and, therefore, are valid at the time when fwd 1st byte

and/or bwd_lst_byte reach the Prediction Filters
Adder.

3) The signals are examined a clock before data arrives anyway.

cast officialities bits out at

-

.<u>=</u>

B.12.4 Prediction Adder and FIFO

The prediction adder (padder) forms the predicted frame by adding the data from the prediction filters to the error data. To compensate for the delay from the input through the address generator, DRAM interface and prediction filters, the error data passes through a 256 word FIFO (sfifo) before reaching padder.

The CODING_STANDARD, PREDICTION_MODE and DATA Tokens are decoded to determine when a predicted block is being formed. The 8-bit prediction data is added to the 9-bit two's complement error data in the DATA Token. The result is restricted to the range 0 to 255 and passes to the next block. Note that this data restriction also applies to all intra-coded data, including JPEG.

The prediction adder of the present invention also includes a mechanism to detect mismatches in the data arriving from the FIFO and the prediction filters. In theory, the amount of data from the filters should exactly correspond to the number of DATA Tokens from the FIFO which involve prediction. In the event of a serious malfunction, however, padder will attempt to recover.

The end of the data blocks from the FIFO and filters are marked, respectively, by the in_extn and fl_last inputs. Where the end of the filter data is detected before the end of the DATA Token, the remainder of the token continues to the output unchanged. If, on the other hand, the filter block is longer than the DATA Token, the input is stalled until all the extra filter data has been accepted and discarded.

There is no snooper in either the FIFO or the prediction adder, as the chip can be configured to pass data from the token input port directly to these blocks, and to pass their output directly to the token output port.

B.12.5 Write and Read Rudders

35 B.12.5.1 The Write Rudder (wrudder)

The Write Rudder passes all tokens coming from the Prediction Adder on to the Read Rudder. It also passes all

data blocks in I or P pictures in MPEG, and all data blocks in H.261 to the DRAM interface so that they can be written back into the external frame stores under the control of the Address Generator. All the primary functionality is contained within one two-wire interface stage, although the write-back data passes through a snooper on its way to the DRAM interface.

The Write Rudder decodes the following tokens:

Token Name	Function in Write Rudder
CODING_STANDARD	Write-back is inhibited for JPEG streams.
PICTURE_TYPE	Write-back only occurs in I and P frames, not 8 frames.
DATA	Only the data within DATA tokens is written back.

B.12.3 Tokens Decoded by the Write Rudder

After the DATA Token header has been detected, all data bytes are output to the DRAM Interface. The end of the DATA Token is detected by in_extn going low and this causes a flush signal to be sent to the DRAM Interface swing buffer. In normal operation, this will align with the point when the swing buffer would swing anyway, but if the DATA Token does not contain 64 bytes of data this provides a recovery mechanism (although it is likely that the next few output pictures would be incorrect).

10

B.12.5.2 The Read Rudder (rrudder)

The Read Rudder of the present invention has three functions, the two major ones relating to picture sequence reordering in MPEG:

- 1) To insert data which has been read-back from the external frame store into the token stream at the correct places.
 - 2) To reorder picture header information in I and P pictures.
- 3) To detect the end of a token stream by detecting the FLUSH token (see Section B.12.1, "Top Fork").

The structure of the Read Rudder is illustrated in Figure 150. The entire block is made from standard two-wire interface technology. Tokens in the input interface latches are decoded and these decodes determine the operation of the block:

Token Name	" Function in Read Rudder			
FLUSH	Signals to Top Fork.			
CODING_STANDARD	Reordering is inhibited if the coding standard is not MPEG.			
SEQUENCE_START	The read-back data for the first picture of a reordered sequence is invalid			
PICTURE_START	Signals that the current output FIFO must be swapped (1 or 2 pictures)			
	The first of the picture header tokens.			
PICTURE_END	All lokens above the picture layer are allowed through			
TEMPORTAL_REFERENCE	The second of the picture header tokens.			
PICTURE_TYPE	The third of the picture header tokens.			
DATA	When reordering, the contents of DATA tokens are represed with			
	recidered data.			

Table B.12.4 Tokens decoded by the Read Rudder

The state of the s

5

15

The reorder function is turned on via the Microprocessor Interface, but is inhibited if the coding standard is not MPEG, regardless of the state of the register. The same MPI register controls whether the Address Generator generates a reorder address and thus, reorder is an output from this block. To understand how the Read Rudder works, consider the input and output control logic separately, bearing in mind that the sequence of tokens is as follows:

- · CODING STANDARD
- 10 · SEQUENCE START
 - · PICTURE START
 - · TEMPORAL_REFERENCE
 - · PICTURE TYPE
 - ·Picture containing DATA Tokens and other tokens
- 15 · PICTURE_END

٠ . . .

· PICTURE START

PICTURE_START

B.12.5.2.1 Input Control Logic

From the power-up, all tokens pass into FIFO 1 (called the current input FIFO) until the first PICTURE_TYPE token for an I or P picture is encountered. FIFO 2 then becomes the current input FIFO and all input is directed to it until the next PICTURE_TYPE for an I or P picture is encountered and FIFO 1 becomes the current input FIFO again. Within I and P pictures, all tokens between PICTURE_TYPE and PICTURE_END, except DATA Tokens, are discarded. This is to prevent motion vectors, etc. from being associated with the wrong pictures in the reordered stream, where they would have no meaning.

A three-bit code is put into the FIFO, along with the token stream, to indicate the presence of certain token headers. This saves having to perform token decoding on the output of the FIFOs.

35 B.12.5.2.2 Output Control Logic

From the power-up, tokens are accepted from FIFO 1 (called the *current output FIFO*) until a picture start code

Contract to the second

15

is encountered, after which FIFO 2 becomes the current output FIFO. Referring back to Section B.12.5.2.1, it can be seen that at this stage the three picture header tokens, PICTURE_START, TEMPORAL_REFERENCE and PICTURE_START are retained in FIFO 1. The current output FIFO is swapped every time a picture start code is encountered in an I or P frame. Accordingly, the three picture header tokens are stored until the next I or P frame, at which time they will become associated with the correctly reordered data. B pictures are not reordered and, hence, pass through without any tokens being discarded. All tokens in the first picture, including PICTURE_END are discarded.

During I and P pictures, the data contained in DATA Tokens in the token stream is replaced by reordered data from the DRAM Interface. During the first picture, "reordered" data is still present at the reordered data input because the Address Generator still requests the DRAM Interface to fetch it. This is considered garbage and is discarded.

The state of the s

SECTION B.13 The DRAM Interface

B.13.1 Overview

5

15

20

25

30

35

In the present invention, the Spatial Decoder, Temporal Decoder and Video Formatter each contain a DRAM Interface block for that particular chip. In all three devices, the function of the DRAM Interface is to transfer data from the chip to the external DRAM and from the external DRAM into the chip via block addresses supplied by an address generator.

The DRAM Interface typically operates from a clock which is asynchronous to both the address generator and to the clocks of the various blocks through which data is passed. This asynchronism is readily managed, however, because the clocks are operating at approximately the same frequency.

Data is usually transferred between the DRAM Interface and the rest of the chip in blocks of 64 bytes (the only exception being prediction data in the Temporal Decoder). Transfers take place by means of a device known as a "swing buffer". This is essentially a pair of RAMs operated in a double-buffered configuration, with the DRAM interface filling or emptying one RAM while another part of the chip empties or fills the other RAM. A separate bus which carries an address from an address generator is associated with each swing buffer.

Each of the chips has four swing buffers, but the function of these swing buffers is different in each case. In the Spatial Decoder, one swing buffer is used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, one swing buffer is used to write Intra or Predicted picture data to the DRAM, the second to read Intra or Predicted data from the DRAM and the other two to read Intra or Predicted data from the DRAM and the other two to read forward and backward prediction data. In the Video Formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data

representation of the restriction of

10

15

20

25

30

35

from the DRAM, one of each of Luminance (Y) and the Red and Blue color difference data (Cr and Cb respectively).

The operation of the generic features of the DRAM Interface is described in the Spatial Decoder document. The following section describes the features peculiar to the Temporal Decoder.

B.13.2 The Temporal Decoder DRAM Interface

As mentioned in section B.13.1, the Temporal Decoder has four swing buffers: two are used to read and write decoded Intra and Predicted (I and P) picture data and these operate as described above. The other two are used to fetch prediction data.

In general, prediction data will be offset from the position of the block being processed as specified by motion vectors in x and y. Thus, the block of data to be fetched will not generally correspond to the block boundaries of the data as it was encoded (and written into the DRAM). This is illustrated in Figures 151 and 25, where the shaded area represents the block that is being formed. The dotted outline shows the block from which it is being predicted. The address generator converts the address specified by the motion vectors to a block offset (a whole number of blocks), as shown by the big arrow, and a pixel offset, as shown by the little arrow.

In the address generator, the frame pointer, base block address and vector offset are added to form the address of the block to be fetched from the DRAM. If the pixel offset is zero, only one request is generated. If there is an offset in either the x or y dimension, then two requests are generated - the original block address and the one either immediately to the right or immediately below. With an offset in both x and y, four requests are generated. For each block which is to be fetched, the address generator calculates start and stop addresses parameters and passes these to the DRAM interface. The use of these start and stop addresses is best illustrated by an example, as outlined below.

· principal of the contract of

Consider a pixel offset of (1, 1), as illustrated by the shaded area in Fig. 152 and Fig. 26. The address generator makes four requests, labelled A through D in the figure. The problem to be solved is how to provide the required sequence of row addresses quickly. The solution is to use "start/stop" technology, and this is described below.

Consider block A in Figure 152. Reading must start at position (1, 1) and end at position (7, 7). Assume for the moment that one byte is being read at a time (i.e. an 8 bit DRAM Interface). The x value in the coordinate pair forms the three LSBs of the address, the y value the three MSBs. The x and y start values are both 1, giving the address 9. Data is read from this address and the x value is The process is repeated until the x value incremented. reaches its stop value. At this point, the y value is incremented by 1 and the x start value is reloaded, giving an address of 17. As each byte of data is read, the x value is again incremented until it reaches its stop value. The process is repeated until both x and y values have reached their stop values. Thus, the address sequence of 9, 10, 11, 12, 13, 14, 15, 17, ..., 23, 25, ..., 31, 33, ..., ..., 57, ..., 63 is generated.

In a similar manner, the start and stop coordinates for block B are: (1, 0) and (7, 0), for block C: (0,1) and (0,7), and for block D: (0, 0) and (0, 0).

The next issue is where this data should be written. Clearly, looking at block A, the data read from address 9 should be written to address 0 in the swing buffer, the data from address 10 to address 15 in the swing buffer, and so on. Similarly, the data read from address 8 in block B should be written to address 15 in the swing buffer and the data from address 16 into address 15 in the swing buffer. This function turns out to have a very simple implementation as outlined below.

Consider block A. At the start of reading, the swing buffer address register is loaded with the inverse of the stop value, the y inverse stop value forming the 3 MSBs and

or brief Built Hun. In think have have the brief and have built and

10

15

20

25

30

35

=

. princes a product care or a

15

20

=

the x inverse stop value forming the 3 LSBs. In this case, while the DRAM Interface is reading address 9 in the external DRAM, the swing buffer address is zero. The swing buffer address register is then incremented as the external DRAM address register is incremented, as illustrated in Table B.13.1:

Table B.13.1 Illustration of Prediction Addressing

Ext DRAM Address	Swing Buff Address	Ext DRAM Ad.	Swing Buff Ad.	
	Comy San Accress	(Binary)	(Binary)	
9 = y-start, x-start	$0 = \overline{y - stop}, \overline{x - stop}$	001 001	000 000	
10	1	111 110	000 001	
11	2	001 011	000 010	
15	6	001 111	000 110	
17 = y+1, x-start	8 = y+1, x-stop	010 001	001 000	
18	9	010 010	001 001	

The discussion thus far has centered on an 8 bit DRAM Interface. In the case of a 16 or 32 bit interface, a few minor modifications must be made. First, the pixel offset vector must be "clipped" so that it points to a 16 or 32 bit boundary. In the example we have been using, for block A, the first DRAM read will point to address 0, and data in addresses 0 through 3 will be read. Next, the unwanted data must be discarded. This is performed by writing all data into the swing buffer (which must now be physically bigger than was necessary in the 8 bit case) and reading with an offset. When performing MPEG half-pel interpolation, 9 bytes in x and/or y must be read from the In this case, the address generator DRAM Interface. provides the appropriate start and stop addresses and some additional logic in the DRAM Interface is used, but there

the contract of the contract o



is no fundamental change in the way the DRAM Interface operates.

The final point to note about the Temporal Decoder DRAM Interface is that additional information must be provided to the prediction filters to indicate what processing is required on the data. This consists of the following:

- ·a "last byte" signal indicating the last byte of a transfer (of 64, 72 or 81 bytes)
- an H.261 flag

10

- ·a bidirectional prediction flag
- ·two bits to indicate the block's dimensions (8 or 9 bytes in x and y)
- a two bit number to indicate the order of the blocks The last byte flag can be generated as the data is read out of the swing buffer. The other signals are derived 15 from the address generator and are piped through the DRAM Interface so that they are associated with the correct block of data as it is read out of the swing buffer by the prediction filter block.

The second of th

and the second s

min in a collection of the second of the

SECTION B.14 UPI Documentation

B.14.1 Introduction

This document is intended to give the reader appreciation of the operation of the microprocessor interface in accordance with the present invention. interface is basically the same on both the SPATIAL DECODER and the Temporal Decoder, the only difference being the number of address lines.

The logic described here is purely the microprocessor internal logic. The relevant schematics are: 10

UPI

UPI101

UPI102

DINLOGIC

15 DINCELL

UPIN

TDET

NONOVRLP

WRTGEN

20 READGEN

25

30

VREFCKT

The circuits UPI, UPI101, UPI102 are all the same except that the UPI01 has a 7 bit address input with the 8th bit hardwired to ground, while the other two have an 8 bit address input.

Input/Output Signals

The signals described here are a list of all the inputs and outputs (defined with respect to the UPI) to the UPI module with a note detailing the source or destination of these signals:

· NOTRSTInputGlobal chip reset, active low, from Pad

ElInputEnable signal 1, active low, from the Pad Input Driver (Schmitt).

35 E21nputEnable signal 2, active low, from the Pad Input Driver (Schmitt).

RNOTWInputRead not Write signal from the Pad Input

requirements of the second of

5

15

30

35

Driver (Schmitt).

ADDRIN[7:0]InputAddress bus signals from the Pad Input Drivers (Schmitt).

NOTDIN[7:0]Inputlnput data bus from the Input Pad Drivers of the Bi-directional Microprocessor Data pins (TTLin).

INT RNOTWOutputThe Internal Read not Write signal to the internal circuitry being accessed by microprocessor interface (See memory map).

INT ADDR[7:0]OutputThe Internal Address Bus to all 10 the circuits being accessed by the microprocessor interface (See memory map).

INTDBUS[7:0]Input/OutputThe Internal Data bus to all the circuits being accessed by the microprocessor interface (See the memory map) and also the microprocessor data output pads. The internal Data bus transfers data which is the inverse to that on the pins of the chip.

READ STROutputAn is an internal timing signal which indicates a read of a location in the device memory map.

20 WRITE STROutputAn is an internal signal which indicates a write of a location in the internal memory map.

TRISTATEDPADOutputAn is an internal signal which connects to the microprocessor data output pads which indicates that they should be tristate.

General Comments: 25

The UPI schematic consists of 6 smaller modules: NONOVRLP, UPIN, DINLOGIC, VREFCKT, READGEN, WRTGEN. should be noted from the overall list of signals that there are no clock signals associated with the microprocessor interface other than the microprocessor bus timing signals which are asynchronous to all the other timing signals on Therefore, no timing relationship should be the chip. assumed between the operation of the microprocessor and the rest of the device other than those that can be forced by For example, stopping of the System external control. microprocessor while accessing the externally interface on a test system.

Therefore a transfer to the contract of the contr

Mood Build flows I stands the control of control of the control of

5

10

15

20

25

30

The other implication of not having a clock in the UPI is that some internal timing is self timed. That is, the delay of some signals is controlled internally to the UPI block.

The overall function of the UPI is to take the address, data and enable and read/write signals from the outside world and format them so that they can drive the internal circuits correctly. The internal signals that define access to the memory map are INT_RNOTW_INT_ADDR[...], INTDBUS[...] and READ_STR and WRITE_STR. The timing relationship of these signals is shown below for a read cycle and a write cycle. It should be noted that although the datasheet definition and the following diagram always shows a chip enable cycle, the circuit operation is such that the enable can be held low and the address can be cycled to do successive read or write operations. function is possible because of the address transition circuits.

Also, the presence of the INT_RNOTW and the READ_STR, WRITE_STR does reflect some redundancy. It allows internal circuits to use either a separate READ_STR and WRITE_STR (and ignore INT_RNOTW) or to use the INT_RNOTW and a separate Strobe signal (Strobe signal being derived from OR of READ_STR and WRITE_STR).

The internal databus is precharged High during a read cycle and it also has resistive pullups so that for extended periods when the internal data bus is not driven it will default to the OXFF condition. As the internal databus is the inverse of the data on the pins, this translates to 0x00 on the external pins, when they are enabled. This means that, if any external cycle accesses a register or a bit of a register which is a hole in the memory map, then the output data id determinate and is Low.

Circuit Details:

35 <u>UPIN</u> -

This circuit is the overall change detect block. It contains a sub-circuit called TDET which is a single bit

....

The state of the s

change detect circuit. UPIN has a TDET module for each address bit and rnotw and for each enable signal. UPIN also contains some combinatorial logic to gate together the outputs of the change detect circuits. This gating generates the signals:

TRAN- which indicates a transition on one of the input signals, and

UPD-DONE- which indicates that transitions have been completed and a cycle can be performed.

10 CHIP_EN- which indicates that the chip has been selected. \underline{TDET} -

This is the single bit change detect circuit. It consists of a 2 latches, and 2 exclusive OR gates. The first latch is clocked by the signal SAMPLE and the second by the signal UPDATE. These two non-overlapping signals come from the module NONOVRLP. The general operation is such that an input transition causes a CHANGE which, in turn, causes a SAMPLE. All input changes while SAMPLE is high are accepted and when input changes cease then CHANGE goes low and SAMPLE goes low which causes UPDATE to go high which then transfers data to the output latch and indicates UPD DONE.

NONOVRLP-

15

20

25

Ĩ.

Ţ.

This circuit is basically a non-overlapping clock generator which inputs TRAN and generates SAMPLE and UPDATE. The external gating on the output of UPDATE stops UPDATE from going high until a write pulse has been completed.

DINLOGIC-

This module consists of eight instances of the data input circuit DINCELL and some gating to drive the TRISTATEPAD signal. This indicates that the output data port will only drive if Enable1 is low, Enable2 is low, RnotW is high and the internal read_str is high.

35 DINCELL-

This circuit consists of the data input latch and a tristate driver to drive the internal databus. Data from

The second section is a second second section of the second section is a second section of the second section of the second section is a second section of the section of the second section of the section of



the input pad is latched when the signal DATAHOLD is high and when both Enablel and Enable2 are low. The tristate driver drives the internal data bus whenever the internal signal INT_RNOTW is low. The internal databus precharge transistor and the bus pullup are also included in this module.

WRTGEN-

10

15

20

25

30

This module generates the WRITE_STR, and the latch signal DATAHOLD for the data latches. The write strobe is a self timed signal, however, the self time delay is defined in the VREFCKT. The output from the timing circuit RESETWRITE is used to terminate the WRITE_STR signal. It should be noted that the actual write pulse which writes a register only occurs after an access cycle is concluded. This is because the data input to the chip is sampled only on the back edge of the cycle. Hence, data is only valid after a normal access cycle has concluded.

READGEN-

This circuit, as its name suggests, generates the READ STR and it also generates the PRECH signal which is used to precharge the internal databus. The PRECH signal is also a self timed signal whose period is dependant on VREFCKT and also on the voltage on the internal databus. The READ STR is not self timed, but lasts from the end of the precharge period until the end of the cycle. precharge circuitry uses inverters with their transfer characteristic biased so that they need a voltage of approximately 75% of supply before they invert. circuit quarantees that the internal bus is correctly precharged before a READ STR begins. In order to stop a PRECH pulse tending to zero width if the internal bus is already precharged, the timing circuit guarantees a minimum, width via the signal RESETREAD.

VREFCKT-

The VREFCKT is the only circuit which controls the self timing of the interface. Both the delays, 1/Width of WRITE_STR and 2/Width of PRECH, are controlled by a current

through a P transistor. The gate on this P transistor is controlled by a signal VREF and this voltage is set by a diffusion resistor of 25K ohm.

SECTION C.1 Overview

C.1.1. Introduction

5

10

15

20

25

30

35

The structure of the image Formatter, in accordance with the present invention, is shown in Figure 155. There are two address generators, one for writing and one for reading, a buffer manager which supervises the two address generators and which provides frame-rate conversion, a data processing pipeline, including both vertical and horizontal unsamplers, color-space conversion and gamma correction, and a final control block which regulates the output of the processing pipeline.

C.1.2 Buffer manager

Tokens arriving at the input to the Image Formatter are buffered in the FIFO and then transferred into the buffer manager. This block detects the arrival of new pictures and determines the availability of a buffer in which to store each picture. If there is a buffer available, it is allocated to the arriving picture and its index is transferred to the write address generator. If there is no buffer available, the incoming picture will be stalled until one becomes available. All tokens are passed on to the write address generator.

Each time the read address generator receives a VSYNC signal from the display system, a request is made to the buffer manager for a new display buffer index. If there is a buffer containing complete picture data, and that picture is deemed ready for display, then that buffer's index will be passed to the display address generator. If not, the buffer manager sends the index of the last buffer to be displayed. At start-up, zero is passed as the index until the first buffer is full.

A picture is ready for display if its number (calculated as each picture is input) is greater than or equal to the picture number which is expected at the display (presentation number) given the encoding frame rate. The expected number is determined by counting picture clock

· process of the first of the

pulses, where picture clock can be generated either locally by the clock dividers, or externally. This technology allows frame-rate conversion (e.g., 2-3 pull-down). External DRAM is used for the buffers, which can be

External DRAM is used for the buffers, which can be either two or three in number. Three are necessary if frame-rate conversion is to be effected.

C.1.3 Write Address Generator

The write address generator receives tokens from the buffer manager and detects the arrival of each new DATA Token. As each DATA Token arrives, the address generator calculates a new address for the DRAM interface for storing the arriving block. The raw data is then passed to the DRAM interface where it is written into a swing buffer. Note that DRAM addresses are block addresses, and pictures in the DRAM or organized as rasters of blocks. Incoming picture data, however, is actually organized sequences of macroblocks, so the address generation algorithm must take into account line-width (in blocks) offsets for the lower rows of blocks within the macroblock.

The arrival buffer index provided by the buffer manager is used as an address offset for the whole of the picture being stored. Furthermore, each component is stored in a separate area within the specified buffer, so component offsets are also used in the calculation.

25 C.1.4 Read Address Generator

The Read Address Generator (dispaddr) does not receive or generate tokens, it generates addresses only. In response to a VSYNC, it may, depending on field_info, read_start, sync_mode, and lsb_invert, request a buffer index from the buffer manager. Having received an index, it generates three sets of addresses, one for each component, for the current picture to be read in raster order. Different setups allow for: interlaced/progressive display and/or data, vertical unsampling, and field synchronization (to an interlaced display). At the lower level, the Read Address Generator converts base addresses into a sequence of block addresses and byte counts for each

the took that there is the test of the tes

10

15

30

of the three components that are compatible with the page structure of the DRAM. The addresses provided to the DRAM interface are page and line addresses along with block start and block end counts.

5 C.1.5 Output Pipeline

10

15

20

25

30

35

Data from the DRAM interface feeds the output pipeline. The three component streams are first vertically interpolated, then horizontally interpolated. the interpolators, the three components should be of equal ratios (4:4:4), and are passed through the color-space converter and color lookup tables/gamma correction. output interface may hold the streams at this point until the display has reached an HSYSC. Thereafter, output controller directs the three components into one, two or three 8-bit buses, multiplexing as necessary.

C.1.6 Timing Regimes

are basically two principal timing regimes associated with the Image Formatter. First, there is a system clock, which provides timing for the front end of the chip (address generators and buffer manager, plus the front end of the DRAM interface). Second, there is a pixel clock which drives all the timing for the back end (DRAM interface output, and the whole of the output pipeline).

Each of the two aforementioned clocks drives a number of on-chip clock generators. The FIFO, buffer manager and read address generator operate from the same clock $(D\Phi)$ with the write address generator using a similar, Data is clocked into the DRAM separate clock (WΦ). interface on an internal DRAM interface clock, (out). WΦ and outΦ are all generated from syscik.

Read and write addresses are clocked in the DRAM interface by the DRAM interface's own clock.

Data is read out of the DRAM interface on bifR&, and is transferred to the section of the output pipeline named "bushy ne" (north-east - by virtue of its physical location) which operates on clocks denoted by NE. section of the pipeline from the gamma RAMs onward 15

• pre-reg

clocked on a separate, but similar, clock (R ϕ). bifR ϕ , NE ϕ and R ϕ are all derived from the pixel clock, pixin.

For testing, all of the major interfaces between blocks have either snoopers or super-snoopers attached. This depends on the timing regimes and the type of access required. Block boundaries between separate, but similar timing regimes have retiming latches associated therewith.

1.0

SECTION C.2 Buffer Management

C.2.1. Introduction

5

10

15

20

25

30

35

The purpose of the buffer management block, in accordance with the present invention, is to supply the address generators with indices identifying any of either two or three external buffers for writing and reading of picture data. The allocation of these indices is influenced by three principal factors, each representing the effect of one of the timing regimes in operation. These are the rate at which picture data arrives at the input to Image Formatter (coded data rate), the rate at which data is displayed (display data rate), and the frame rate of the encoded video sequence (presentation rate).

C.2.2 Functional Overview

A three-buffer system allows the presentation rate and the display rate to differ (e.g., 2-3 pulldown), so that frames are either repeated or skipped as necessary to achieve the best possible sequence of frames given the timing constraints of the system. Pictures which present some difficulty in decoding may also be accommodated in a similar way, so that if a picture takes longer than the available display time to decode, the previous frame will be repeated while everything else "catches up". In a two-buffer system, the three timing regimes must be locked - it is the third buffer which provides the flexibility for taking up the slack.

The buffer manager operates by maintaining certain status information associated with each external buffer. This includes flags indicating if the buffer is in use, if it is full of data, or ready for display, and the picture number within the sequence of the picture currently stored in the buffer. The presentation number is also recorded, this being a number which increments every time a picture clock pulse is received, and represents the picture number which is currently expected for display based on the frame rate of the encoded sequence.

- -

=

15

20

25

An arrival buffer (a buffer to which incoming data will be written) is allocated every time a PICTURE_START token is detected at the input. This buffer is then flagged as IN_USE. On PICTURE_END, the arrival buffer will be deallocated (reset to zero) and the buffer flagged as either FULL or READY depending on the relationship between the picture number and the presentation number.

The display address generator requests a new display buffer, once every vsync, via a two-wire interface. If there is a buffer flagged as READY, then that will be allocated to display by the buffer manager. If there is no READY buffer, the previously displayed buffer will be repeated.

Each time the presentation number changes, it is detected and every buffer containing a complete picture is tested for READY-ness by examining the relationship between its picture number and the presentation number. Buffers are considered in turn. When any of the buffers are deemed to be READY, this automatically cancels the READY-ness of any buffer which was previously flagged as READY. The previous buffer is then flagged as EMPTY. This works because later picture numbers are stored, by virtue of the allocation scheme, in the buffers that are considered later.

TEMPORAL_REFERENCE tokens in H.261 cause a buffer's picture number to be modified if skipped pictures in the input stream are indicated. This feature, although envisioned, is not currently included, however. Similarly, TEMPORAL-REFERENCE tokens in MPEG have no effect.

A FLUSH token causes the input to stall until every buffer is either EMPTY or has been allocated as the display buffer. Thereafter, presentation number and picture number are reset and a new sequence can commence.

=

1 p B

15

25

30

35

C.2.3 Architecture

C.2.3.1 Interfaces

C.2.3.1.1. Interface to bm front

All data is input to the buffer manager from the input FIFO, bm_front. This transfer takes place via a two-wire interface, the data being 8 bits wide plus an extension bit. All data arriving at the buffer manager is guaranteed to be a complete token. This is a necessity for the continued processing of presentation numbers and display buffer requests in the event of significant gaps in the data upstream.

C.2.3.1.2 Interface to waddrgen

Tokens (8 bit data, 1 bit extension) are transferred to the write address generator via a two-wire interface. The arrival buffer index is also transferred on the same interface, so that the correct index is available for address generation at the same time as the PICTURE_START token arrives at waddrgen.

C.2.3.1.3 Interface to dispaddr

The interface to the read address generator comprises two separate two-wire interfaces which can be considered to act as "request" and "acknowledge" signals, respectively. Single wires are not adequate, however, because of the two-wire-based state machines at either end.

The sequence of events normally associated with the dispaddr interface is as follows. First, dis-paddr invokes a request in response to a vsync from the display device by asserting the drq valid input to the buffer manager. Next, when the buffer manager reaches an appropriate point in its state machine, it will accept the request and go about allocating a buffer to be displayed. Thereafter, disp valid wire asserted, the buffer index is transferred, and this is typically accepted immediately by Furthermore, there is an additional wire dispaddr. associated with this last two-wire interface (rst_fld) which indicates that the field number associated with the current index must be reset regardless of the previous

field number.

C.2.3.1.4 Microprocessor Interface

The buffer manager block uses four bits of microprocessor address space, together with the 8-bit data bus and read and write strobes. There are two select signals, one indicating user-accessible locations and the other indicating test locations which should not require access under normal operating conditions.

C.2.3.1.5 Events

10

15

25

30

The buffer manager is capable of producing two different events, index found and late arrival. The first of these is asserted when a picture arrives and its PICTURE_START extension byte (picture index) matches the value written into the BU_BM_TARGET_IX register at setup. The second event occurs when a display buffer is allocated and its picture number is less than the current presentation number, i.e., the processing in the system pipeline up to the buffer manager has not managed to keep up with the presentation requirements.

20 C.2.3.1.6 Picture Clock

In the present invention, picture clock is the clock signal for the presentation number counter and is either generated on-chip or taken from an external source (normally the display system). The buffer manager accepts both of these signals and selects one based on the value of pclk_ext (a bit in the buffer manager's control register). This signal also acts as the enable for the pad picoutpad, so that if the Image Formatter is generating its own picture clock, this signal is also available as an output from the chip.

C.2.3.2. Major Blocks

The following sections describe the various hardware blocks that make up the buffer manager schematic (bmlogic).

C.2.3.2.1 Input/Output block (bm input)

This module contains all of the hardware associated with the four two-wire interfaces of the buffer manager (input and output data, drq valid/accept and disp valid/accept).

- III - II

H ...

-

15

20

25

30

35

W.

The input data register is shown, together with some token decoding hardware attached thereto. The signal vheader at the input to bm_tokdec is used to ensure that the token decoder outputs can only be asserted at a point where a header would be valid (i.e., not in the middle of a token. The rtimd block acts as the output data registers, adjacent to the duplicate input data registers for the next block in the pipeline. This accounts for timing differences due to different clock generators. Signals go and ngo are based on the AND of data valid, accept and not stopped, and are used elsewhere in the state machine to indicate if things are "bunged up" at either the input or the output.

The display index part of this module comprises the two-wire interfaces together with equivalent "go" signals as for data. The rst_fld bit also happens here, this being a signal which, if set, remains high until disp_valid has been high for one cycle. Thereafter, it is reset. In addition, rst_fld is reset after a FLUSH token has caused all of the external buffers to be flagged either as EMPTY or IN_USE by the display buffer. This is the same point at which both picture numbers and presentation number are reset.

There is a small amount of additional circuitry associated with the input data register which appears at the next level up the hierarchy. This circuitry produces a signal which indicates that the input data register contains a value equal to that written into BU_BM_TARGIX and it is used for event generation.

C.2.3.2.2 Index block (bm index)

The Index block consists mainly of the 2-bit registers denoting the various strategic buffer indices. These are arr_buf, the buffer to which arriving picture data is being written, disp_buf, the buffer from which picture data is being read for display, and rdy_buf, the index of the buffer containing the most up to date picture which could be displayed if a buffer was requested by dispaddr. There is also a register containing buf_ix, which is used as a

dillity differ manners on two con-

The Hallman

buffer.

This

register

pointer to a

incremented ("D" input to mux) to cycle through the buffers examining their status, or which gets assigned the value of one of arr_buf, disp_buf or rdy_buf when the status needs changing. All of these registers (ph0 versions) accessible from the microprocessor as part of the test address space. Old_ix is just a re-timed version of buf_ix and is used for enabling buffer status and picture number registers in the bm stus block. Both buf_ix and old ix are decoded into three signals (each can hold the value 1 to 3) which are output from this block. Other outputs indicate whether buf_ix has the same value as either arr buf or disp_buf, and whether either of rdy_buf and disp_buf have the value zero. Zero is not a reference to a buffer. merely indicates that there is no arrival/display/ready buffer currently allocated.

Arr_buf and disp_buf are enabled by their respective twowire interface output accept registers.

Additional circuitry at the bmlogic level is used to determine if the current buffer index (buf ix) is equal to the maximum index in use as defined by the value written into the control register at setup. A "1" in the control register indicates a three-buffer system, and a indicates a two-buffer system.

25 C.2.3.2.3 Buffer Status

general

5

10

15

20

30

35

The main components in the buffer status are status and picture number registers for each buffer. Each of the groups of three is a master-slave arrangement where the slaves are the banks of three registers, and the master is a single register whose output is directed to one of the slaves (switched, using register enables, by old ix). One of the possible inputs to the master is multiplexed between the different slave outputs (indexed by buf_ix at the bmlogic level). Buffer status, which is decoded at the bmlogic level, for use in the state machine logic can take any of the values shown in Table C.2.1, or recirculate its previous value. Picture number can take the previous value

10

15

or the previous value incremented by one (or one plus delta, the difference between actual and expected temporal reference, in the case of H.261). This value is supplied by the 8-bit adder present in the block. The first input to this adder is this pnum, the picture number of the data currently being written.

Buffer Status	Value	
EMPTY	00	
FULL	01	
READY	. 10	
IN_USE	11	

Table C.2.1 Buffer Status Values

This needs to be stored separately (in its own master-slave arrangement) so that any of the three buffer picture number registers can be easily updated based on the current (or previous) picture number rather than on their own previous picture number (which is almost always out of date). This pnum is reset to -1 so that when the first picture arrives it is added to the output from the adder and, hence, the input to the first buffer picture number register, is zero.

Note that in the current version, delta is connected to zero because of the absence of the temporal reference block which should supply the value.

20 C.2.3.2.4 Presentation Number

The 8-bit presentation number register has an associated presentation flag which is used in the state machine to indicate that the presentation number has changed since it

e pero con

10

15

25

30

35

was last examined. This is necessary because the picture clock is essentially asynchronous and may be active during any state, not just those which are concerned with the presentation number. The rest of the circuitry in this block is concerned with detecting that a picture clock pulse has occurred and "remembering" this fact. way, the presentation number can be updated at a time when it is valid to do so. A representative sequence of events is shown in Figure 156. The signal incr_prn goes active the cycle after the re-timed picture clock rising edge, and persists until a state is entered during which presentation number can be modified. This is indicated by the signal en prnum. The reason for only allowing presentation number to be updated during certain states is because it is used to drive a significant amount of logic, including a standard-cell, not-very-fast 8-bit adder to provide the signal rdyst. It must, therefore, be changed only during states in which the subsequent state does not use the result.

C.2.3.2.5 Temporal Reference 20

The temporal reference block in accordance with the present invention, has been omitted from the current embodiment of the Image Formatter, but its operation is described here for completeness.

The function of this block is to calculate delta, the difference between the temporal reference value received in a token in an H,261 data stream, and the "expected" temporal reference (one plus the previous value). allows frames to be skipped in H.261. Temporal reference tokens are ignored in all non-H.261 streams. calculated value is used in the status block to calculate The effect of omitting picture numbers for the buffers. the block from bmlogic is that picture numbers will always be sequential in any sequence, even if the H.261 stream indicates that some should be skipped.

The main components of the block (visible in the schematic bm_tref) are registers for tr, exptr and delta.

inger operate

10

In the invention, tr is reset to zero and loaded, when appropriate, from the input data register. Similarly, exptr is reset to -1, and is incremented by either 1 or delta during the sequence of temporal reference states. In addition, delta is reset to zero and is loaded with the difference between the other two registers. registers are reset after a FLUSH token. The adder in this block is used for calculation of both delta and exptr, i.e., a subtract and an add operation, respectively, and is controlled by the signal delta_calc.

10

15

20

C.2.3.2.6 Control Registers (bm uregs)

Control registers for the buffer manager reside in the block bm_uregs. These are the access bit register, setup register (defining the maximum number of external buffers, and internal/external picture clock), and the target index register. The access bit is synchronized as expected. The signals stopd_0, stopd_1 and nstopd_1 are derived form the OR of the access bit and the two event stop bits. Upi address decoding for all of bmlogic is done by the block bm_udec, which takes the lower 4 bits of the upi data bus together with the 2 select signals from the Image Formatter top-level address decode.

C.2.3.2.7 Controlling State Machine

The state machine logic originally occupied its own block, bm_state. For code generation reasons, however, it has now been flattened and resides on sheet 2 of the bmlogic schematic.

The main sections of this logic are the same. This includes the decoding, the generation of logic signals for the control of other bmlogic blocks, and the new state encoding, including the flags from ps and from fl which are used to select routes through the state machine. There are separate blocks to produce the mux control signals for bm stus and bm index.

Signals in the state machine hardware have been given simple alphabetic names for ease of typing and reference. They are all listed in Table C.2.2, together with the logic expressions which they represent. They also appear as comments in the behavioral M. description of bmlogic (bmlogic.M).

The state of the s

Table C.2.2 Signal Names Used in the State Machine

591

Signal Name	· Logic Expression			
s	ST_PICTURE_ENO.(ix==arr).!rdytstoar			
Т	ST_PICTURE_END.(ix==arr).rdytst.(rdy==0).oar			
U	ST_PICTURE_ENO.(ix==arr).rdytst.(rdy!=0) oar			
~	ST_PICTURE_ENO.:loar			
RorVV	ST_PICTURE_END.!((ix==arr).oar)			
V	ST_TEMP_REF0.ivr.oar			
w ·	ST_TEMP_REF0.!(ivr.oar)			
×	ST_OUTPUT_TAIL.ivr.oar			
FF	ST_OUTPUT_TAIL.ivr.oar.lin_extn			
Y	ST_OUTPUT_TAIL.!(ivr.oar)			
GG	ST_OUTPUT_TAIL.!(ivr.oar).in_extn			
DD	ST_FLUSH.(ix==max).((bstate==VAC)+((bstate==USE).(ix==disp))			
Z	ST_FLUSH.(ix!=max).((bstate==VAC)+((bstate==USE).(ix==disp))			
DDorEE	!((bstate==VAC)+((bstate==USE).(ix==disp))+(ix==max)			
AA	ST_ALLOC.(bstate==VAC).oar			
88	ST_ALLOC.(bstate!=VAC).(ix==max)			
CC	ST_ALLOC.(bstate!=VAC).(ix!=max)			
υυ	ST_ALLOC.!oar			

Table C.2.2 Signal names Used in the State Machine

C.2.3.2.8 Monitoring Operation (bminfo)

In the present invention, the module, bminfo, is included so that buffer status information, index values and presentation number can be observed during simulations. It is written in M and produces an output each time one of its inputs changes.

C.2.3.3 Register Address Map

The buffer manager's address space is split into two areas, user-accessible and test. There are, therefore, two separate enable wires derived from range decodes at the top-level. Table C.2.3 shows the user-accessible registers, and Table C.2.4 shows the contents of the test space.

-

Register Name	Accress	, 5 ts .	=ese	Function
	İ	! !	:	<u> </u>
		 	State	
BU_BM_ACCESS	1 0x10	(0)	1	Access bit for buffer manager
BU_SM_CTL0	Ox1:	(0)	1	Max bufish, 1->3 buffers.0->2
		[1]		External picture clock select
BU_SM_TARGET_IX	0x:2	[3 0]	0x0	For detecting arrival of picture
BU_SM_PRES_NUM	0x13	[7 0]	0x00	Presentation number
BU_BM_THIS_PNUM	0x14	[[7 기	0xFF	Current picture number
BU_BM_PIC_NUM0	Cx15	[F 0]	поле	Picture number in buder 1
BU_BM_PIC_NUM1	Ox16	[7.0]	none	Picture number in buffer 2
BU_BM_PIC_NUM2	0x17	[7.0]	none	Picture number in buffer 3
BU_BM_TEMP_REF	0x18	[40]	0x00	Temporal reference from stream

Table C.2.3 User-Accessible Registers

Register Name	Address	3.62	Reset State	Function
BU_SM_PRES_FLAG	C×80	, [O]	0	Presentation "ag
BU_BM_EXP_TR	0x81	[4 0]	0xFF	Expected temporal reference
BU_BM_TR_DELTA	0x82	[4 0]	0x00	Celta
SU_BM_ARA_IX	CBxO	[1.0]	0x0	Arrival buffer index
BU_BM_OSP_IX	0x84	[1.0]	0x0	Display buffer index
BU_BM_ROY_IX	0x85	[1.0]	0x0	Ready buffer index
BU_BM_BSTATE3	0x85	[: 0]	0x0	Buffer 3 status
BU_BM_BSTATE2	0x87	[1-0]	0x0	Buffer 2 status
BU_BM_BSTATE!	0xe8	[(1 0)	0x0	Buffer 1 status
BU_SM_INDEX	0x89	[1 0]	0x0	Current buffer index
BU_BM_STATE	Ox8A	[4 0]	0x00	Buffer manager state
BU_BM_FROMPS	0188	[0]	0x0	From PICTURE_START
				flag
BU_SM_FROMFL	0x8C) (C)	0×0	From FLUSH_TOKEN flag

Table C.2.4 Test Registers

0 1

1 1 1 1

C.2.4 Operation of The State Machine

There are 19 states in the buffer manager's state machine, as detailed in Table C.2.5. These interact as shown in Figure 157, and also as described in the behavioral description bmlogic.M.

State	Value
PRESO	0×00
PRES1	0x10
ERROR	0x1F
TEMP_REFO	0x04
TEMP_REF1	0x05
TEMP_REF2	0×06
TEMP_REF3	0x07
ALLOC	0x03
NEW_EXP_TR	0x0D
SET_ARR_IX	0x0E
NEW_PIC_NUM	0x0F
FLUSH	0x01
DRQ	0x0B
TOKEN	0x0C
OUTPUT_TAIL	0x08
VACATE_RDY	0x17
USE_RDY	0x0A
VACATE_DISP	0x09
PICTURE_END	0x02

Table C.2.5 Buffer States

n n n



The reset state is PRESO, with flags set to zero such that the main loop circulated initially.

C.2.4.2 The Main Loop

5

10

15

20

25

The main loop of the state machine comprises the states shown in Figure 153 (high-lighted in the main diagram -States PRESO and PRES1 are concerned with Figure 152). detecting a picture clock via the signal presflg. cycles are allowed for the tests involved since they all depend on the value of rdyst, the adder output signal described in C.2.3.2.4. If a presentation flag is detected, all of the buffers are examined for possible 'readiness', otherwise the state machine just advances to state DRQ. Each cycle around the PRESO-PRES1 loop examines a different buffer, checking for full and ready conditions. If these are met, the previous ready buffer (if one exists) is cleared, the new ready buffer is allocated and its This process is repeated until all status is updated. buffers have been examined (index == max buf) and the state then advances. A buffer is deemed to be ready for display when any of the following is true:

```
(pic_num>pres_num)&&((pic_num - pres_num)>=128)

or
(pic_num<pres_num)&&((pres_num - pic_num)<=128)

or
pic_num == pres_num
```

State DRQ checks for a request for a display buffer (drq_valid_reg && disp_acc_reg). If there is no request the state advances (normally to state TOKEN - as will be described later). Otherwise, a display buffer index is issued as follows. If there is no ready buffer, the previous index is re-issued or, if there is no previous display buffer, a null index (zero) is issued. If a buffer

15

20

25

30

35

is ready for display, its index is issued and its state is updated. If necessary, the previous display buffer is cleared. The state machine then advances as before.

State TOKEN is the typical option for completing the main loop. If there is valid input and the output is not stalled, tokens are examined for strategic values (described in later sections), otherwise control returns to state PRESO.

Control only diverges from the main loop when certain conditions are met. These are described in the following sections.

C.2.4.3 Allocating The Ready Buffer Index

If during the PRESO-PRESI loop a buffer is determined to be ready, any previous ready buffer needs to be vacated because only one buffer can be designated ready at any time. State VACATE_RDY clears the old ready buffer by setting its state to VACANT, and it resets the buffer index to 1 so that when control returns to the PRESO state, all buffers will be tested for readiness. The reason for this is that the index is by now pointing at the previous ready buffer (for the purpose of clearing it) and there is no record of our intended new ready buffer index. It is necessary, therefore, to re-test all of the buffers.

C.2.4.4 Allocating The Display Buffer Index

Allocation of the display buffer index takes place either directly from state DRQ (state USE_RDY) or via state VACATE_DISP which clears the old display buffer state. The chosen display buffer is flagged as IN_USE, the value of rdy_buf is set to zero, and the index is reset to 1 to return to state DRQ. Moreover, disp_buf is given the required index and the two-wire interface wires (disp_valid and drq_acc) are controlled accordingly. Control returns to state DRQ only so that the decision between states TOKEN, FLUSH and ALLOC does not need to be made in state USE_RDY.

C.2.4.5 Operation when PICTURE_END Received

On receipt of a PICTURE_END token, control transfers from

15

20

25

30

35

state TOKEN to state PICTURE_END where, if the index is not already pointing at the current arrival buffer, it is set to point there so that its status can be updated. Assuming both out_acc_reg and en_full are true, status can be updated as described below. If not, control remains in state PICTURE_END until they are both true. The en_full signal is supplied by the write address generator to indicate that the swing buffer has swung, i.e., the last block has been successfully written and it is, therefore, safe to update the buffer status.

The just-completed buffer is tested for readiness and given the status either FULL or READY depending on the result of the test. If it is ready, rdy_buf is given the value of its index and the set_la_ev signal (late arrival event) is set high (indicating that the expected display has got ahead in time of the decoding). The new value of arr_buf now becomes zero and, if the previous ready buffer needs its status clearing, the index is set to point there and control moves to state VACATE_RDY. Otherwise, the index is reset to 1 and control returns to the start of the main loop.

When a PICTURE_START token arrives during state TOKEN, the flag from ps is set, causing the basic state machine loop to be changed such that state ALLOC is visited instead of state TOKEN. State ALLOC is concerned with allocating an arrival buffer (into which the arriving picture data can be written), and cycles through the buffers until it finds one whose status is VACANT. A buffer will only be allocated if out_acc_reg is high since it is output on the data two-wire interface. Accordingly, cycling around the loop will continue until this is indeed the case. suitable arrival buffer has been found, the index is allocated to arr buf and its status is flagged as IN_USE. Index is set to 1, the flag from ps is reset, and the state is set to advance to NEW EXP TR. A check is made on the

10

15

20

25

30

35

picture's index (contained in the word following the PICTURE_START) to determine if it is the same as targ_ix (the target index specified at setup) and, if so, set_if+_ev (index found event) is set high.

The three states NEW_EXP_TR, SET_ARR_IX and NEW_PIC_NUM set up the new expected temporal reference and picture number for the incoming data. The middle state just sets the index to be arr_buf so that the correct picture number register is updated (note that this_pnum is also updated). Control then proceeds to state OUTPUT_TAIL which outputs data (assuming favorable two-wire interface signals) until a low extension is encountered. At this point, the main loop is re-started. This means that whole data blocks (64 items) are output, in between which, there are no tests for presentation flags or display requests.

C.2.4.7 Operation When FLUSH Received

A FLUSH token in the data stream indicates that sequence information (presentation number, picture number, rst_fld) should be reset. This can only occur when all of the data leading up to the FLUSH has been correctly processed. Accordingly, it is necessary, having received a FLUSH, to monitor the status of all of the buffers until it is certain that all frames have been handed over to the display, i.e., all but one of the buffers have status EMPTY, and the other is IN_USE (as the display buffer). At that point, a "new sequence" can safely be used.

When a FLUSH token is detected in state TOKEN, the flag from_fl is set, causing the basic state machine loop to be changed such that state FLUSH is visited instead of state TOKEN. State FLUSH examines the status of each buffer in turn, waiting for it to become VACANT or IN_USE as display. The state machine simply cycles around the loop until the condition is true, then increments its index and repeats the process until all of the buffers have been visited. When the last buffer fulfills the condition, presentation number, picture number, and all of the temporal reference registers assume their reset values rst fld is set to 1.

DE 1 11

15

The flag from_fl is reset and the normal main loop operation is resumed.

C.2.4.8 Operation When TEMPORAL_REFERENCE Received

When a TEMPORAL_REFERENCE token is encountered, a check is made on the H.261 bit and, if set, the four states TEMP_REF0 to TEMP_REF3 are visited. These perform the following operations:

TEMP_REF0:temp_ref=in_data reg;

TEMP_REF1:delta=temp_ref-exp_tr;index=arr buf;

10 TEMP_REF2:exp_tr=delta+exp tr;

TEMP_REF3:pic_num[i]=this_pnum+delta;index=1.

C.2.4.9 Other Tokens and Tails

State TOKEN passes control to state OUTPUT_TAIL in all cases other than those outlined above. Control remains here until the last word of the token is encountered (in_extn_reg is low) and the main loop is then re-entered.

C.2.5 Applications Notes

C.2.5.1 State Machine Stalling Buffer Manager Input

This requirement repeatedly check for the "asynchronous" timing events of picture clock and display buffer request. 20 The necessity of having the buffer manager input stalled during these checks means that when there is a continuous supply of data at the input to the buffer manager, there will be a restriction on the data rate through the buffer 25 manager. A typical sequence of states may be PRESO, PRESI, TOKEN, OUTPUT_TAIL, each, with the exception of OUTPUT_TAIL, lasting one cycle. This means that for each block of 64 data items, there will be an overhead of 3 cycles during which the input is stalled (during states 30 PRESO, PRES1 and DRQ) thereby slowing the write rate by 3/64 or approximately 5%. This number may occasionally increase to up to 13 cycles of overhead when auxiliary branches of the state machine are executed under worst-case conditions. Note that such large overheads will only apply 35 on a once-per-frame basis.

C.2.5.2 Presentation Number Behavior During An Access

The particular embodiment of the bm pres illustrated by

1. 11

ΗI

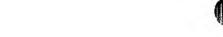
и ирт ти

the schematic shown in C.2.3.2.4 means that presentation number free-runs during upi accesses. If presentation number is required to be the same when access is relinquished as it was when access was gained, this can be effected by reading presentation number after access is granted, and writing it back just before it is relinquished. Note that this is asynchronous, so it may be desirable to repeat the accesses several times to further ensure effectiveness.

10 C.2.5.3 H261 Temporal Reference Numbers

The module bm_tref (not shown) should be included in the bmlogic. The H.261 temporal reference values are correctly processed by directing delta input from the bmtref to the bm_stus module. The delta input can be tied to zero if the frames are always sequential.

and the property of the second



SECTION C.3 Write Address Generation

C.3.1 Introduction

5

10

15

20

25

30

The function of the write address generation hardware, in accordance with the present invention, is to produce block addresses for data to be written away to the buffers. This takes account of buffer base addresses, the component indicated in the stream, horizontal and vertical sampling within a macroblock, picture dimensions, and coding standard. Data arrives in macroblock form, but must be stored so that lines may be retrieved easily for display.

C.3.2 Functional Overview

Each time a new block arrives in the data stream (indicated by a DATA token), the write address generator is required to produce a new block address. necessary to produce the address immediately, because up to 64 data words can be stored by the DRAM interface (in the swing buffer) before the address is actually needed. means that the various address components can be added to a running total in successive cycles, and thus, hence obviating the need for any hardware multipliers. macroblock counter function is effected by storing strategic terminal values and running counts register file, these being the operands for comparisons and conditional updates after each block address calculation.

Considering the picture format shown in Figure 161, expected address sequences can be derived for both standard and H.261-like data streams. These are shown below. Note that the format does not actually conform to the H.261 specification because the slices are not wide enough (3 macroblocks rather than 11) but the same "half-picture-width-slice" concept is used here for convenience and the sequence is assumed to be "H.261-type". Data arrives as full macroblocks, 4:2:0 in the example shown, and each component is stored in its own area of the specified

35 buffer.

Standard address sequence: 000,001,00C,00D,100,200; 002,003,00E,00F,101,201; 004,005,010,011,102,202; 006,007,012,013,103,203; 008,009,014,015,104,105; coA,00B,016,017,105,205;

018,019,024,025,106,107;

01A.01B,026......

080,081,08C,08D,122,222; 082,083,08E,08F,123,223;

H261-type sequence:

000,001,00C,00D,100,200;

002,003,00E,00F,101,201;

004,005,010,011,102,202;

018,019,024,025,106,107;

01A,01B,026,027,107,207;

01C,01D,028,029,108,208;

030,031,03C,03D,10C,20C,

032,033,03E,03F,10D,20D;

034,035,040,041,10E,20E;

006,007,012,013,103,203;

008,009,014,015,104,105;

00A,00B,016,017,105,205;

01E,01F,02A,02B,109,209;

020,021,02C,02D,10A,20A;

15

022,023,02E,02F,10B,20B;

036,037,042,043,10F,20F;

038,039,044,045,110,210;

C3A,03B,046,047,111,211;

048,049,054,055,112,212;

04A,04B,056.....

•••••

06A,06B,076,077,11D,21D;

07E,07F,08A,08B,121,221;

080,081,08C,08D,122,222;

082,083,08E,08F,123;223;

C.3.3 Architecture

C.3.3.1 Interfaces

C.3.3.1.1 Interface to buffer manager

The buffer manager outputs data and the buffer index directly to the write address generator. This is performed under the control of a two-wire-interface. In some ways, it is possible to consider the write address generator block as an extension of the buffer manager because the two are very closely linked. They do, however, operate from two separate (but similar) clock generators.

C.3.3.1.2 Interface to dramif

The write address generator provides data and addresses for the DRAM interface. Each of these has their own two-wire-interface, and the dramif uses each of them in different clock regimes. In particular, the address is clocked into the dramif on a clock which is not related to the write address generator clock. It is, therefore, synchronized at the output.

C.3.3.1.3 Microprocessor Interface

The write address generator uses three bits of microprocessor address space together with 8-bit data bus and read and write strobes. There is a single select bit for register access.

1.11

15

20

35

C.3.3.1.4 Events

The write address generator is capable of producing five different events. Two are in response to picture size information appearing in the data stream (hmbs and vmbs), and three are in response to DEFINE_SAMPLING tokens (one event for each component.

C.3.3.2 Basic Structure

The structure of the write address generator is shown in the schematic waddrgen.sch. It comprises a datapath, some controlling logic, and snoopers and synchronization.

C.3.3.2.1 The Datapath (bwadpath)

The datapath is of the type described in Chapter C.5 of this document, comprising an 18-bit adder/subtractor and register file (see C.3.3.4), and producing a zero flag (based on the adder output) for use in the control logic.

C.3.3.2.2 The Controlling Logic

The controlling logic of the present invention consists of hardware to generate all of the register file load and drive signals, the adder control signals, the two-wire-interface signals, and also includes the writable control registers.

C.3.3.2.3 Snoopers and Synchronization

Super snoopers exist on both the data and address ports. Snoopers in the datapaths, controlled as super-snoopers from the zcells. The address has synchronization between the write address generator clock and the dramif's "clk" regime. Syncifs are used in the zcells for the two-wire interface signals, and simplified synchronizers are used in the datapath for the address.

30 C.3.3.3 Controlling Logic and State Machine

C.3.3.3.1 Input/Output Block (wa inout)

This block contains the input and two output two-wire interfaces, together with latches for the input data (for token decode) and arrival buffer index (for decoding four ways).

C.3.3.3.2 Two Cycle Control Block (wa fc)

The flag fc (first cycle) is maintained here and

11 1

1,0

10

30

35

indicates whether the state machine is in the middle of a two-cycle operation (i.e., an operation involving an add).

C.3.3.3.3. Component Count (wa comp)

Separate addresses are required for data blocks in each component, and this block maintains the current component under consideration based on the type of DATA header received in the input stream.

C.3.3.3.4 Modulo-3 Control (wa mod3)

When generating address sequences for H.261 data streams, it is necessary to count three rows of macroblocks to half way along the screen (see C.3.2). This is effected by maintaining a modulo-3 counter, incremented each time a new row of macroblocks is visited.

C.3.3.3.5 Control Registers (wa uregs)

Module wa uregs contains the setup register and the 15 coding standard register - the latter is loaded from the data stream. The setup register uses 3 bits: QCIF (1sb) and the maximum component expected in the data stream (bits 1 and 2). The access bit also resides in this block (synchronized as usual), with the "stopped" bits being 20 derived at the next level up the hierarchy (walogic) as the OR of the access bit and the event stop Microprocessor address decoding is done by the block wa udec which takes read and write strobes, a select wire, 25 and the lower two bits of the address bus.

C.3.3.3.6 Controlling State Machine (wa state)

The logic in this block is split into several distinct areas. The sate decode, new state encode, derivation of "intermediate" logic signals, datapath control signals (drivea, driveb, load, adder controls and select signals), multiplexer controls, two-wire-interface controls, and the five event signals.

C.3.3.3.7 Event Generation

The five event bits are generated as a result of certain tokens arriving at the input. It is important that, in each case, the entire token is received before any events are generated because the event service routines perform

calculations based on the new values received. For this reason, each of the bits is delayed by a whole cycle before being input to the event hardware.

C.3.3.4 Register Address Map

There are two sets of registers in the write address generator block. These are the top-level setup type registers located in the standard cell section, and keyholed datapath registers. These are listed in Table C.3.1 and C.3.2, respectively.

Register Name	Address	Bits	Reset	Function
			State	
BU_WADDR_COD_STD	0x4	2	0	Cod std from data stream
BU_WADDR_ACCESS	0×5	1	0	Access bit
BU_WADDR_CTL1	0x6	3	0	max component(2:1) and
				CCIF(0)
BU_WA_ADDR_SNP2	0x80	8		snooper on the write
BU_WA_ADDR_SNP1	0x81	8		address generator
BU_WA_ADDR_SNP0	0x82	8		address c/p.
BU_WA_DATA_SNP1	0x84	8		snooper on data output of
BU_WA_DATA_SNP0	0x85	8		WA

Table C.3.1 Top-Level Registers

5

	Keynole	عرد	Comments
Keyhole Register Name	Address	210	l Commens
BU_WADCR_BUFFERO_BASE_MSB	0×95	2	Must be
BU_WADOR_BUFFERO_BASE_MID	C×86	3	Loaded
BU_WADDR_BUFFERO_BASE_LSB	0x87	9	
BU_WADOR_BUFFER1_BASE_MSB	Gx89	2	Must be
BU_WADDR_BUFFER!_BASE_MID	0x3a	3	Loaced
BU_WACOR_BUFFER1_BASE_LSB	Gx8b	8	
BU_WADDR_SUFFER2_BASE_MSB	0x8d	2	Must be
BU_WADDR_BUFFER2_BASE_MID	0x8e	8	Loaded
BU_WACOR_BUFFER2_BASE_LSB	0×81	8	1
BU_WADDR_CCMP0_HMBADDR_MSB	0x91	2	Test only
BU_WADDR_COMPO_HMBADDR_MID	0x92	8	!
BU_WACCA_COMPO_HMBADDA_LSB	Cx93	9	1
BU_WADDR_COMP1_HMBADDR_MSB	0x95	2	Test on y
BU_WACCR_COMP1_HMBACCR_MID	0x96	8	
BU_WADDR_COMP1_HMBADOR_LSB	0x97	8	<u> </u>
BU_WADDR_COMP2_HMBADDR_MSB	0x99	2	Test only
BU_WADDR_CCMP2_HMBADDR_MID	Cx9a	8	
BU_WADDR_COMP2_HMBADDR_LSB	0x9b	8	
BU_WADDR_COMPO_VMBADDR_MSB	0x9d	2	Test only
BU_WADDR_COMP0_VMBADDR_MID	0x9e	8	
SU_WADDR_COMPO_VMBADDR_LSB	0x91	В	
BU_WADDR_COMP1_VMBADDR_MSB	0xa1	2	Test only
BU_WADDR_COMP1_VMBADDR_MID	0xa2	8	
BU_WADDR_COMP1_VMBADDR_LSB	0xa3	В	
BU_WADDR_COMP2_VMBADDR_MSB	0×a5	2	Test only
BU_WADDR_COMP2_VMBADDR_MID	0xa6	8	_
BU_WADDR_COMP2_VMBADDR_LSB	0xa7	8	
BU_WADDR_VBADDR_MSB	0xa9	2	Test only
SU_WADOR_VBADOR_MID	Oxaa	8	<u></u>
BU_WADDR_VBADDR_LSB	Oxab	8	i

Table C.3.2 Image Formatter Address Generator Keyhole



Keyhole Register Name	Keynola Address	වාප	Comments
BU_WADDR_COMPO_HALF_WIDTH_IN_BLOCKS_MSB	Cxad	2	Mustice
BU_WADDR_COMPO_HALF_WIDTH_IN_BLOCKS_MID	Oxae	8	Loaded
BU_WADDR_COMPO_HALF_WIOTH_IN_BLOCKS_LSB	Oxaf	8	
BU_WADDR_COMP1_HALF_WIDTH_IN_BLOCKS_MSB	0xb1	2	Must be
BU_WACCR_COMP1_HALF_WIDTH_IN_BLOCKS_MIO	Gxb2	8	Loaded
BU_WACOR_COMP1_HALF_WIOTH_IN_BLOCKS_LSB	0xb3	8	
BU_WADDR_COMP2_HALF_WIDTH_IN_BLOCKS_MSB	0xb5	2	Must be
SU_WADCA_COMPZ_HALF_WIDTH_IN_SLOCKS_MID	0×66	8	Loaced
BU_WACOR_COMP2_HALF_WIDTH_IN_BLOCKS_LSB	0xb7	8	
SU_WADOR_HB_MSB	0xb9	2	Test only
OIM_BH_ROCKW_UB	Oxba	8	- ! _;
BU_WACOR_HB_LSB	Oxbb	а	
BU_WADDR_COMPO_OFFSET_MSB	Oxbd	2	Must be
BU_WADOR_COMPO_OFFSET_MID	8	Loaded	
BU_WADDR_COMPO_OFFSET_LSB	0xbf	3	- !
BU_WADDR_COMP1_OFFSET_MSB	0xc1	2	Must be
BU_WADDR_COMP1_OFFSET_MID	ìз	Loased	
BU_WADDR_COMP1_OFFSET_LSB	Cxc3	a	Ī
BU_WADOR_COMP2_OFFSET_MSB	Cxc5	2	Must be
BU_WADDR_COMP2_OFFSET_MID	0xc6	8	Loaded
BU_WADDR_COMP2_OFFSET_LSB	0xc7	8]
BU_WACOR_SCRATCH_MSB	0xc9	2	Test only
BU_WADDR_SCRATCH_MID	Oxca	8	
BU_WADDR_SCRATCH_LS8	Охсъ	8	1
BU_WADDR_MBS_WIDE_MS3	0xcd	2	Must be
3U_WADDA_MBS_WIDE_MID	0xce	8	Loaced
BU_WADDR_MBS_WIDE_LSB	Oxci	8	
BU_WADOR_MBS_HIGH_MSB	0xd1	2	Must be
BU_WADDA_MBS_HIGH_MID	0xd2	8	Loaded
BU_WADOR_MBS_HIGH_LSB	0xd3	8	

Table C.3.2 Image Formatter Address Generator Keyhole

the second secon

Keynole Register Name

Keynoie

Address

Bits :

Comments

Table C.3.2 Image formatter Address Generator Keyhole

a great to the

dillo			
	feytoe	3.:3	Comments
Keynole Register Name	Address	j	
BU_WADDR_COMPO_LAST_MB_ROW_MSB	0x105	2	Must be
BU_WADDR_COMPO_LAST_MB_ROW_MID	0x106	8	Loaded
BU_WADDR_COMPO_LAST_MB_ROW_LSB	0x107	8	
BU_WADDR_COMP1_LAST_MB_ROW_MSB	0x109	2	Must be
SU_WADDR_COMP1_LAST_MB_ROW_MID	0x10a	8	Loaded
BU_WACOR_COMP1_LAST_MB_ROW_LSB	0x10b	8	!
BU_WADDR_COMP2_LAST_MB_ROW_MSB	0x10d	2	Must be
BU_WADDR_COMP2_LAST_MB_ROW_MID	0x10e	8	Loaded
BU_WADDR_COMP2_LAST_MB_ROW_LSB	0x10f	8	
BU_WADDR_COMPO_HBS_MSB	0x111	2	Must be
BU_WADDR_COMP0_HBS_MID	0x112	8	Loaded
BU_WADDR_COMPO_HBS_LSB	0x113	8	
BU_WADDR_COMP1_HBS_MSB	0x115	2	Must be
BU_WADDR_COMP1_HBS_MID	0x116	8	Loaded
BU_WADDR_COMP1_HBS_LSB	0x117	8	
BU_WADDR_COMP2_HBS_MSB	0x119	2	Must be
BU_WADDR_COMP2_HBS_MID	Ox11a	8	Loaded
BU_WADDR_COMP2_HBS_LSB	0x11b	8	
BU_WADDR_COMP0_MAXHB	0x11f	2	Must be
BU_WADDR_COMP1_MAXHB	0x123	2	Loaced
BU_WADDR_COMP2_MAXHB	0x127	2	
BU_WADDR_COMP0_MAXVB	0x12b	2	Must be
BU_WADDR_COMP1_MAXVB	0x12!	2	Loaded
BU_WADDR_COMP2_MAXVB	0x133	2	

Table C.3.2 Image Formatter Address Generator Keyhole

The keyhole registers fall broadly into two categories. Those which must be loaded with picture size parameters prior to any address calculation, and those which contain running totals of various (horizontal and vertical) block and macroblock counts. The picture size parameters may be loaded in response to any of the interrupts generated by the write address generator, i.e., when any of the picture size or sampling tokens appear in the data stream. Alternatively, if the picture size is known prior to receiving the data stream, they can be written just after reset. Example setups are given in Sectionr C.13, and the picture size parameter registers are defined in the next section.

10

5

366610..

10

15

25

30

C.3.4 Programming the Write Address Generator

The following datapath registers must contain the correct picture size information before address calculation can proceed. They are illustrated in Figure 162.

- 1)WADDR_HALF_WIDTH_IN_BLOCKS: this defines the half width, in blocks, of the incoming picture.
 - 2) WADDR_MBS_WIDE: this defines the width, in macroblocks, of the incoming picture.
 - 3) WADDR_MBS_HIGH: this defines the height, in macroblocks, of the incoming picture.
 - 4) WADDR_LAST_MB_IN_ROW: this defines the block number of the top left hand block of the last macroblock in a single, full-width row of macroblocks. block numbering starts at zero in the top left corner of the left-most macroblock, increases across the frame with each block and subsequently with each following row of blocks within the macroblock row.
- 5) WADDR_LAST_MB_IN_HALF_ROW: this is similar to the previous item, but defines the block number of the top left block in the last macroblock in a half-width row of macroblocks.
 - 6) WADDR_LAST_ROW_IN_MB: this defines the block number of the left most block in the last row of blocks within a row of macroblocks.
 - 7) WADDR_BLOCKS_PER_MB_ROW: this defines the total number of blocks contained in a single, full-width row of macroblocks.
 - 8) WADDR_LAST_MB_ROW: this defines the top left block address of the left-most macroblock in the last row of macroblocks in the picture.
- 35 9) WADDR_HBS: this defines the width in blocks of the incoming picture.
 - 10) WADDR MAXHB: this defines the block number

the property of the contract o



of the right-most block in a row of blocks in a single macroblock.

- 11) WADDR_MAXVB: this defines the height-1, in blocks, of a single macroblock.
- In addition, the registers defining the organization of the DRAM must be programmed. These are the three buffer base registers, and the n component offset registers, where n is the number of components expected in the data stream (it can be defined in the data stream; and can be 1 minimum and 3 maximum).

Note that many of the parameters specify block numbers or block addresses. This is because the final address is expected to be a block address, and the calculation is based on a cumulative algorithm.

The screen configuration illustrated in Figure 162 yields the following register values:

1)WADDR_HALF_WIDTH_IN_BLOCKS = 0x16

2)WADDR_MBS_WIDE = 0x16

3)WADDR_MBS_HIGH = 0x12

4)WADDR_LAST_MB_IN_ROW = 0x2A

5)WADDR_LAST_MB_IN_HALF_ROW = 0x14

6)WADDR_LAST_ROW_IN_MB = 0x2C

7)WADDR_BLOCKS_PER_MB_ROW = 0x58

8)WADDR_LAST_MB_ROW = 0x5D8

9)WADDR_HBS = 0x2C

10)WADDR_MAXVB = 1

11)WADDR_MAXH3 = 1

C.3.5 Operation of The State Machine

There are 19 states in the buffer manager's state machine, as detailed in Table C.3.3. These interact as shown in Figure 164, and also as described in the behavioral description, bmlogic.M.

State	Value
IDLE	0x00
DATA	0x10
CODING_STANDARD	0x0C
HORZ_MBS0	0x07
HORZ_MBS1	0x06
VERT_MBS0	0x0B
VERT_MBS1	0x0A
OUTPUT_TAIL	0x08
нв	0x11
MB0	0x1D
MB1	0x12
мв2	0x1E
мвз	0x13
MB4	0x0E
мв5	0x14
MB6	0x15
MB4A	0x18

Table C.3.3 Write Address Generator States

and the second s

State -	Value
MB48	0×09
MB4C	0x17
MB40	0x15
ADDRI	*0x19
ADDR2	0x1A
ADDR3	0x18
ADDR4	0×1C
ADDR5	0×03
HSAMP	0×05
VSAMP	0x04
PIC_ST1	0x0f
PIC_ST2	0x01
PIC_ST3	0x02

Table C.3.3 Write Address Generator States

C.3.5.1 Calculation of the Address

The major section of the write address generator state machine is illustrated down the left hand side of Figure 164. On receipt of a DATA token, the state machine moves from state IDLE to state ADDR1 and then through to state ADDR5, from which an 18-bit block address is output with two-wire-interface controls. The calculations performed by the states ADDR1 through to ADDR5 are:

- BU_WADDR_SCRATCH=BU_BUFFERn_BASE +BU_COMPm_OFFSET;
 - BU WADDR SCRATCH=BU WADDR SCRATCH
 - +BU_WADDR_VMBADDR;
 - BU WADDR SCRATCH=BU_WADDR-SCRATCH
- +BU_WADDR_HMBADDR;
 - BU_WADDR_SCRATCH=BU+WADDR_SCRATCH
 - +BU_WADDR_VBADDR;
 - out_addr=BU_WADDR_SCRATCH+BU_WADDR_HB;

The registers used are defined as follows:

20 1) BU_WADDR_VMBADDR: the block address (the top left block) of the left-most macroblock of the row of macroblocks in which the block whose address is being calculated is contained.

Harry other state, man or or or

niiiiiii ii Mara radda radda aa aa aa aa da baa aa waa

- BU WADDR HMBADDR: the block address (top left 2) block) of the top macroblock of the column of macroblocks in which the block whose address is being calculated is contained.
- BU WADDR VBADDR: the block address, within the 3) macroblock row, of the left-most block of the row of blocks in which the block whose address is being calculated is contained.
- BU WADDR HB: the horizontal block number, within 4) the macroblock, of the block whose address is 10 being calculated.
 - BU WADDR SCRATCH: the scratch register used for 5) temporary storage of intermediate results.

Considering Figure 163, and taking, for example, the calculation of the block whose address is 0x62D, 15 following sequence of calculations will take place;

SCRATCH=BUFFERn_BASE+COMPm_OFFSET; (assume 0)

SCRATCH=0+0x5D8;

SCRATCH=0x5D8+0x28;

20 SCRATCH=0x600+0x2C;

block address=0x62C+1=0x62D;

The contents of the various registers are illustrated in the Figure.

C.3.5.2 Calculation of New Screen Location Parameters

When the address has been output, the state machine 25 continues to perform calculations in order to update the various screen location parameters described above. states HB and MBO through to MB6 do the calculations, transferring control at some point to state DATA from which the reminder of the DATA Token is output. 30

These states proceed in pairs, the first of a pair calculating the difference between the current count and its terminal value and, hence, generating a zero flag. The second of the pair either resets the register or adds a fixed (based on values in the setup registers derived from In each case, if the count under screen size) offset. consideration has reached its terminal value (i.e., the

zero flag is set), control continues down the "MB" sequence of states. If not, all counts are deemed to be correct (ready for the next address calculation) and control transfers to state DATA.

Note that all states which involve the use of an addition or subtraction take two cycles to complete (allowing the use of a standard, ripple-carry adder), this being effected by the use of a flag, fc (first cycle) which alternates between 1 and 0 for adder-based states.

All of the address calculation and screen location 10 calculation states allow data to be output assuming favorable two-wire interface conditions.

C.3.5.2.1 Calculations for Standard

(MPEG-style) Sequences

```
The sequence of operations is as follows (in which the
    zero flag is based on the output of the adder):
5
       states HB and MBO:
       scratch = hb - maxhb;
       if(z)
         hb = 0;
       else
10
        (
         hb = hb + 1
         new state = DATA;
        states MB1 and MB2:
        scratch = vb_addr - last_row_in_mb;
15
        if (z)
          vb addr = 0;
        else
          vb addr = vb_addr + width_in_blocks;
20
          new state = DATA;
        states MB3 and MB4:
        scratch = hmb_addr - last_mb_in_row;
25
        if (z)
          hmb_addr = 0;
        else
          hmb_addr = hmb_addr + maxhb;
          new_state = DATA; .
30
        states MB5 and MB6:
        scratch = vmb addr - last_mb_row;
        if (!z)
          vmb_addr = vmb_addr + blocks_per_mb_row;
35
        (vmb_addr is reset after a PICTURE_START token is
     detected, rather than when the end of a picture is inferred
```

The second secon

from the calculations).

C.3.5.2.2 Calculations for H.261 Sequences

```
The sequence for H.261 calculations diverges from the
    standard sequence at state MB4:
       states HB and MBO:-as above
5
       states MB1 and MB2:-as above
       states MB3 and MB4:
       scratch = hmb addr - last_mb_in_row;
        if (z & (mod3==2)) /*end of slice on right of screen*/
10
        (
          hmb addr - 0;
          new_state - MB5;
        else if (z) /*end of row on right of screen*/
15
        (
          hmb_addr = half_width_in_blocks;
          new_state = MB4A;
        )
        else
20
          scratch = hmb_addr - last_mb_in_half_row;
          new-state = MB4B;
        }
```

and the second



```
state MB4A:
vmb_addr = vmb_addr + blocks_per_mb_row;
new_state = DATA;
state (MB4) and MB4B:
(scratch = hmb_addr - last_mb_in_half_row;)
if (z & (mod3==2)) /*end of slice on left of screen*/
  hmb_addr = hmb_addr + maxhb;
  new_state = MB4C;
}
else if (z) /*end of row on left of screen*/
   hmb_addr = 0;
   new_state = MB4A;
 else
   hmb_addr = hmb_addr + maxhb;
   new_state = DATA;
 ì
 states MB4C and MB4D:
 vmb_addr = vmb_addr - blocks_per_mb_row;
 vmb_addr = vmb_addr - blocks_per_mb_row;
 new_state = DATA;
```

states MB5and MB6:- as above

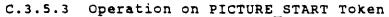
C.3.5.3 Operation on PICTURE_START Token

When a PICTURE_START token is received, control passes to state PIC_ST1 where the vb_addr register (BU_WADDR_VBADDR) is reset to 0. Each of states PIC_ST2 and PIC_ST3 are then visited, once for each component, resetting hmb_addr and vmb_addr respectively. Control then returns, via state OUTPUT_TAIL, to IDLE.

15

20

25



When a PICTURE_START token is received, control passes to state PIC_ST1 where the vb_addr register (BU_WADDR_VBADDR) is reset to 0. Each of states PIC_ST2 and PIC_ST3 are then visited, once for each component, resetting hmb_addr and vmb_addr, respectively. Control then returns, via state OUTPUT_TAIL, to IDLE.

C.3.5.4 Operation on DEFINE_SAMPLING Token

When a DEFINE_SAMPLING token is received, the component register is loaded with the least significant two bits of the input data. In addition, via states HSAMP and VSAMP, the maxhb and maxvb registers for that component are loaded. Furthermore, the appropriate define sampling event bit is triggered (delayed by one cycle to allow the whole token to be written).

C.3.5.5 Operation on HORIZONTAL MBS and VERTICAL MBS

When each of HORIZONTAL_MBS and VERTICAL_MBS arrive, the 14-bit value contained in the token is written, in two cycles, to the appropriate register. The relevant event bit is triggered, delayed by one cycle.

C.3.5.6 Other Tokens

The CODING_STANDARD token is detected and causes the top-level BU_WADDR_COD_STD register to be written with the input data. This is decoded and the nh261 flag (not H261) is hardwired to the buffer manager block. All other tokens cause control to move to state OUTPUT_TAIL, which accepts data until the token finishes. Note, however, that it does not actually output any data.

SECTION C.4 Read Address Generator

C.4.1 Overview

5

10

15

20

The read address generator of the present invention consists of four state machine/datapath blocks. The first, "dline", generates line addresses and distributes them to other three (one for each component) page/block address generators, "dramctls". All blocks are The modes of operation linked by two wire interfaces. include all combinations of interlaced/progressive, first field upper/lower, and frame start on upper/lower/both. The Table C.3.4 shows the names, addresses, and reset states of the dispaddr control registers, and Chapter C.13 gives a programming example for both address generators.

C.4.2 Line Address Generator (dline)

This block calculates the line start addresses for each component. Table C.3.4 shows the 18 bit datapath registers in dline.

Note the distinction between DISP_register_name and ADDR_register_name DISP _name registers are in dispaddr only and means that the register is specific to the display area to be read out of the DRAM. ADDR_name means that the register describes something about the structure of the external buffers.

Operation

The basic operation of dline, ignoring all modes repeats 25 etc. is: if (vsync_start)/* first active cycle of vsync*/ comp = 0DISP_VB_CNT_COMP(comp)=0; 30 LINE(comp)=BUFFER BASE(comp)+0; LINE(comp)=LINE(comp)+DISP_COMP_OFFSET(comp); while (VB CNT COMP[comp] < DISP_VBS_COMP[comp] while (line count[comp]<8) 35

the state of the s

		_
		+
		-
-		•
	2 11 2 11 5	-
	3	=
		٠,
	_	
	~	
-		
	٠,	
-	7	
	=	
		į
	2	
7	-	
. "	۲,	
:		
	_	
7	•	

j

Register Names	Bus	Keynole	Description	Comments .
-3 -1		Address		1
BUFFER_BASEO	A	0x00,01,02,03	Siccx address	These registers
BUFFER_BASE1	А	0x04.05,06,07	of the stan of	must be loaded
BUFFER_BASE2	A	0x08,09,0a,0b	each buffer.	by the uproelore
DISP_COMP_OFFSET0	8	0:24,25,26,27	Offsets from the	operation can
DISP_COMP_OFFSET1	В	0x29,29,2a,2b	buffer base to	begin.
DISP_COMP_OFFSET2	8	Cx2c.2d.2e.2!	where reading	
			begins.	
CISP_VBS_COMPO	В	0x30,31,32,33	Number of	
DISP_VBS_COMP1	В	0x34,35,36,37	vertical blocks	
DISP_VES_COMP2	В	0x38,39,3a,3b	to be read	:
ADDR_HBS_COMP0	В	0x3c,3d,3e,3f	Number of	
ADDR_HBS_COMP1	В	0x40,41,42,43	horizontal	
ACCR_48S_COMP2	8	0x44,45,46,4	blocks IN THE	
			DATA	
UNEO	A	0x0c,0d,0e.0f	Current line	These registers
UNEI	A	0c10,11,12.13	address	are temporary
UNE2	A	0x14,15,16,17		locations used
DISP_VB_CNT_COMPO	A	0x18,19,1a,10	Number of	by dispaddi.
DISP_VB_CNT_COMP1	A	0x1c.1d,1e,1f	vertical blocks	01 0.35840
DISP_VB_CNT_COMP2	A	0:20.21,22,23	remaining to be	Note: All
1			read.	registers are P/
		1	1640-	W from the upi

Table C.3.4 Dispaddr Datapath Registers

There is the contract to a second

C.4.3 Dline Control Registers

The above operation is modified by the dispaddr control registers which are shown in the Table C.4.3 below.

Register Name	Address	Bits	Reset State	Function
LINES_IN_LAST_ROWO	0x08	[2:0]	0x07	These three registers
LINES_IN_LAST_ROW1	0x09	[2:0]	0 x 07	determine the number of
LINES_IN_LAST_ROW2	0x0a	[2:0]	0x07	lines (out of 8) of the fast
				row of blocks to read out
DISPACOR_ACCESS	0x0b	{0}	0x00	Access bit for dispaddr
DISPADOR_CTL0	0x0c	[1:0]	0x0	SYNC_MODE
See below for a detailed		[2]	0x0	READ_START
		[3]	0x1	INTERLACED/PROG
description of these		[4]	0x0	LSB_INVERT
control bits		[7:5]	0x0	LINE_RPT
DISPADDR_CTL1	OxOd	[0]	0x1	COMPOHOLD

Dispaddr Control Registers

TABLE C.4.3 CONTROL REGISTERS

5 C.4.3.1 LINES IN LAST ROW[component]

These three registers determine, for each component, the number of lines in the last row of blocks that are to be read. Thus, the height of the read window may be an arbitrary number of lines. This is a back-up feature since the top, left and right edges of the window are on block boundaries, and the output controller can clip (discard) excess lines.

C.4.3.2 DISPADDR ACCESS

10

15

This is the access bit for the whole of dispaddr. On writing a "1" to this location, dispaddr is halted synchronously to the clocks. The value read back from the access bit will remain "0" until dispaddr has safely halted. Having reached this state, it is safe to perform asynchronous upi accesses to all the dispaddr registers.

CONCERN COMPANY OF THE FIRST CONTRACT OF STREET

10

20

30

35

Note that the upi is actively locked out from the datapath registers until the access bit is "1". In order for access to dispaddr to be achieved without disrupting the current display or datapath operation, access will only given and released under the following circumstances.

Stopping: Access will only be granted if the datapath has finished its current two cycle operation (if it were doing one), and the "safe" signal from the output controller is high. This signal represents the area on the screen below the display window and is programmed in the output controller (not dispaddr). Note: It is, therefore, necessary to program the output controller before trying to gain access to dispaddr.

Starting-Access will only be released when "safe" is high, or during vsync. This ensures that display will not start too close to the active window.

This scheme allows the controlling software to request access, poll until end of display, modify dispaddr, and release access. If the software is too slow and doesn't release the access bit until after vsync, dispaddr will not start until the next safe period. Border color will be displayed during this "lost" picture (rather than rubbish).

C.4.3.3 DISPADDR_CTLO[7:0]

When reading the following descriptions, it is important to understand the distinction between interlaced data and an interlaced display.

Interlaced data can be of two forms. The Top-Level Registers supports field-pictures (each buffer contains one field), and frames (each buffer contains an entire frame - interlaced or not)

DISPADDR_CTL0[7:0] contains the following control bits: SYNC_MODE[1:0]

With an interlaced display, vsyncs referring to top and bottom fields are differentiated by the field_info pin. In this context, field_info = HIGH meaning the top field. These two control bits determine which vsyncs dispaddr will request a new display buffer from the buffer manager and,

percent of the state of the sta

thus, synchronize the fields in the buffers (if the data were interlaced) with the fields on the display:

O: New Display Buffer On Top Field

1:Bottom Field

2:Both Fields

5

10

35

3:Both Fields

At startup, dispaddr will request a buffer from the buffer manager on every vsync. Until a buffer is ready, dispaddr will receive a zero (no display) buffer. When it finally gets a good buffer index, dispaddr has no idea where it is on the display. It may, therefore, be necessary to synchronize the display startup with the correct vsync.

READ START

15 For interlaced displays at startup, this bit determines on which vsync display will actually start. Furthermore, having received a display buffer index, dispaddr may "sit out" the current vsync in order to line up fields on the display with the fields in the buffer.

20 INTERLACED/PROGRESSIVE

0:Progressive

1:Interlaced

In progressive mode, all lines are read out of the display area of the buffer. In interlaced mode, only alternate lines are read. Whether reading starts on the 25 first or second line depends on field info. Note that with (interlaced) field-pictures, the system wants to read all lines from each buffer so the setting of this bit would be progressive. The mapping between field info first/second line start may be inverted by 1sb invert (so 30 named for historical reasons).

LSB INVERT

When set, this bit inverts the field_info signal seen by the line counter. Thus, reading may be started on the correct line of a frame and aligned to the display regardless of the convention adopted by the encoder, the display or the Top-Level Registers.

or a more of the amore care a con-

LINE RPT[2:0]

Each bit, when set, causes the lines of the corresponding component to be read twice (bit 0 affects component 0 etc.). This forms the first part of the vertical unsampling. It is used in the 8 times chroma upsampling required for conversion from QFIF to 601.

COMPOHOLD

10

This bit is used to program the ratio of the number of lines to be read (as opposed to displayed) for component 0 to those of components 1 and 2).

0: Same number of lines, i.e., 4:4:4 data in the buffers.

1: Twice as many component 0 lines, i.e., 4:2:0.

Page/Block Address Generators (dramctls)

15 When passed a line address, these blocks generate a series of page/line addresses and blocks to read along the line. The minimum page width of 8 blocks is always assumed and the resulting outputs consist of a page address, a 3 bit line number, a 3 bit block start, and a 3 bit block stop address. (The line number is calculated by dline and passed through the dramctls unmodified). Thus, to read out 48 pixels of line 5 form page Oxaa starting from the third block from the left (an arbitrary point along an arbitrary line), the addresses passed to the DRAM interface would be:

25 Page = Oxaa

Line = 5

Block start = 2

Block stop = 7

Each of these three machines has 5 datapath registers.

These are shown in Table C.3.4. The basic behavior of each dramctl is:

```
Block start = 2
Block stop = 7
Each of these three machines has 5 datapath registers. These are shown in Table C 3.4
     The basic behaviour of each dramctl is:-
while (true)
{
CNT_LEFT = 0;
GET_A_NEW_LINE_ADDRESS from dline;
BLOCK_ADDR = input_block_addr + 0;
PAGE_ADDR = input_page_addr + 0;
CNT_LEFT = DISP_HBS + 0;
while (CNT_LEFT > BLOCKS_LEFT)
{
BLOCKS_LEFT = 8 - BLOCK_ADDR;
--> output PAGE_ADDR, start=BLOCK_ADDR, stop=7.
PAGE_ADDR = PAGE_ADDR + 1;
BLOCK_ADDR = 0;
CNT_LEFT = CNT_LEFT - BLOCKS_LEFT;
/ Last Page of line 1/
CNT_LEFT = CNT_LEFT + BLOCK_ADDR;
CNT_LEFT = CNT_LEFT - 1;
--> output PAGE_ADDR,start=BLOCK_ADDR,stop=CNT_LEFT
}
```

Table C.3.5 Dramctl(0,1 &2) Datapath Registers

Table C.3.5 Dramctl(0,1 & 2) Datapath Registers

Register Names	Sus	Keyhole Address	Description	Comments
DISP_COMP0_HBS	А	0x48,49,4a,4b	The number of	This register
DISP_COMP1_HBS	A	0x4c,4d,4e,4f	horizontal	must be loaded
DISP_COMP2_HBS	A	0x50,51,52,53	blocks to be	before
			read. c.l.	operation can
			ADDR_H8S	begin.
CNT_LEFTO	A	0x54,55.56,57	Number of	These registers
CNT_LEFT1	A	0x58,59,5a,5b	blocks remaining	are temporary
CNT_LEFT2	A	0x5c,5d,5e,5l	to be read	locations used
PAGE_ADDRO	A	0x60,61,62,63	The address of	by dispaddr.
PAGE_ADDR1	A	0x64.65.66.67	the current	Note: All
PAGE_ADDR2	A	0x68,69,6a,6b	page.	registers are P/
BLCCK_ADDR0	В	0x6c,6d,6e,6f	Current block	W from the upi
BLCCK_ADDR1	В	0x70,71,72,73	address	
BLOCK_ADDR2	5	0x74,75,76,77		1
BLOCKS_LEFT0	В	0x78.79.7a.7b	Blocks left in	
BLOCKS_LEFT!	8	0x7c,7d,7e,7t	current page	
BLOCKS_LEFT2	В	0x80,81,82,83		

Programming

The following 15 dispaddr registers must be programmed before operation can begin.

BUFFER_BASE0,1,2

DISP_COMP_OFFSET0,1,2

DISP_VBS_COMP0,1,2

ADDR_HBS_COMP0,1,2

DISP_COMP0,1,2_HBS

Using the reset state of the dispaddr control registers will give a 4:2n interlaced display with no line repeats synchronized and starting the top field (field info=HIGH). Figure 159, "Buffer 0 Containing a SIF (22 by 18 macroblocks) picture," shows a typical buffer setup for a SIF picture. (This example is covered in more detail in Section C.13). Note that in this example, DISP HBS COMPn is equal to ADDR HBS COMPn and likewise the vertical registers DISP VBS COMPn and the equivalent write address generator register are equal, i.e., the area to be read is the entire buffer.

Windowing with the Read Address Generator

It is possible to program dispaddr such that it will read only a portion (window) of the buffer. The size of the window is programmed for each component by the registers DISP_HBS, DISP_VBS, COMPONENT_OFFSET, and LINES_IN_LAST_ROW. Figure 160, "SIF Component 0 with a display window," shows how this is achieved (for component 0 only).

In this example, the register setting would be:
BUFFER BASEO = 0x00

DISP_COMP_OFFSETO = 0x2D

 $DISP_VBS_COMPO = 0x22$

 $ADDR_HBS_COMPO = Ox2C$

25 DISP HBS COMO = 0x2A

Notes:

•The window may only start and stop on block boundaries. In this example we have left LINES_IN_LAST_ROW equal to 7 (meaning all eight).

This example is not practical with anything other than 4:4:4 data. In order to correspond, the window edges for the other two components could not be on block boundaries.

•The color space converter will hang up if the data it receives is not 4:4:4. This means that these read windows, in conjunction with the upsamplers must be programmed to achieve this.

The third has the the flatter of the back the flatter arms and

5

10

30

30

SECTION C.5 Datapaths for Address Generation

The datapaths used in dispaddr and waddrgen are identical in structure and width (18 bits), only differing in the number of registers, some masking, and the flags returned to the state machine. The circuit of one slice is shown in 5 Figure 165, "Slice Of Datapath,". Registers are uniquely assigned to drive the A or B bus and their use (assignment) is optimized in the controller. All registers are loadable from the C bus, however, not all "load" signals are driven. All operations involving the adder cover two 10 allowing the adder to have ordinary ripple carry. "Two cycle operation of the datapath," shows the timing for the two cycle sum of two registers being loaded back into the "A" bus register. The various flags are 15 "phO"ed within the datapath to allow ccode generation. the same reason, the structure of the datapath schematics is a little unusual. The tristates for all the registers (onto the A and B buses) are in a single block which eliminates the combinatorial path in the cell, therefore, allowing better ccode generation. 20 To gain upi access to the datapaths, the access bit must be set, for without this, the upi is locked out. Upi access is different from read and write:

•Writing: When the access bit is set, all load signals are disabled and one of a set of three byte addressed write strobes driven to the appropriate byte of one of the registers. The upi data bus passes vertically down the datapath (replicated, 2-8-8 bits) and the 18 bit register is written as three separate byte writes

•Reading: This is achieved using the A and B buses. Once again, the access bit must be set. The addressed register is driven onto the A or B bus and a upi byte select picks a byte from the relevant bus and drives it onto the upi bus.

35 As double cycle datapath operations require the A and B buses to retain their values, and upi accesses disrupt



these, access must only be given by the controlling state machine before the start of any datapath operation.

All datapath registers in both address generators are addressed through a 9 bit wide keyhole at the top level address Ox28 (msb) and Ox29 (lsb) for the keyhole, and Ox2A for the data. The keyhole addresses are given in Table C.11.2.

Notes:

10

15

- 1) All address registers in the address generators (dispaddr and waddrgen) contain blocked addresses. Pixel addresses are never used and the only registers containing line addresses are the three LINES_IN_LAST_ROW registers.
- 2) Some registers are duplicated between the address generators, e.g., BUFFER_BASEO occurs in the address space for dispaddr and waddrgen. These are two separate registers which BOTH need loading. This allows display windowing (only reading a portion of the display store), and eases the display of formats other than 3 component video.

20

25

30

35



SECTION C.6 The DRAM Interface

C.6.1 Overview

In the present invention, the Spacial Decoder, Temporal Decoder and Video Formatter each contain a DRAM Interface block for that particular chip. In all three devices, the function of the DRAM Interface is to transfer data from the chip to the external DRAM and from the external DRAM into the chip via block addresses supplied by an address generator.

The DRAM Interface typically operates from a clock which is asynchronous to both the address generator and to the clocks of the various blocks through which data is passed. This asynchronism is readily managed, however, because the clocks are operating at approximately the same frequency.

Data is usually transferred between the DRAM Interface and the rest of the chip in blocks of 64 bytes (the only exception being prediction data in the Temporal Decoder). Transfers take place by means of a device known as a "swing buffer". This is essentially a pair of RAMs operated in a double-buffered configuration, with the DRAM interface filling or emptying one RAM while another part of the chip empties or fills the other RAM. A separate bus which carries an address from an address generator is associated with each swing buffer.

Each of the chips has four swing buffers, but the function of these swing buffers is different in each case. In the Spacial Decoder, one swing buffer is used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, one swing buffer is used to write Intra or Predicted picture data to the DRAM, the second to read Intra or Predicted data from the DRAM and the other two to read forward and backward prediction data. In the Video Formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data from

10

15

20

25

30

35

the DRAM, one for each of Luminance (Y) and the Red and Blue color difference data (Cr and Cb, respectively).

The operation of a generic DRAM Interface is described in the Spacial Decoder document. The following section describes those features of the DRAM Interface, in accordance with the present invention, peculiar to the Video Formatter.

C.6.2 The Video Formatter DRAM Interface

In the video formatter, data is written into the external DRAM in blocks, but read out in raster order. Writing is exactly the same as already described for the Spacial Decoder, but reading is a little more complex.

The data in the Video Formatter external DRAM is organized so that at least 8 blocks of data fit into a single page. These 8 blocks are 8 consecutive horizontal blocks. When rasterizing, 8 bytes need to be read out of each of 8 consecutive blocks and written into the swing buffer (i.e., the same row in each of the 8 blocks).

Considering the top row (and assuming a byte-wide interface), the x address (the three LSBs) is set to zero, as is the y address (3 MSBs). The x address is then incremented as each of the first 8 bytes are read out. At this point, the top part of the address (bit 6 and above - LSB = bit 0) is incremented and the x address (3 LSBs) is reset to zero. This process is repeated until 64 bytes have been read. With a 16 or 32 bit wide interface to the external DRAM, the x address is merely incremented by two or four instead of by one.

The address generator can signal to the DRAM Interface that less than 64 bytes should be read (this may be required at the beginning or end of a raster line) although a multiple of 8 bytes is always read. This is achieved by using start and stop values. The start value is used for the top part of the address (bit 6 and above), and the stop value is compared with this and a signal generated which indicates when reading should stop.

SECTION C.7 Vertical Upsampling

C.7.1 Introduction

Given a raster scan of pixels of one color component at its input, the vertical upsampler in accordance with the present invention, can provide an output scan of twice the height. Mode selection allows the output pixel values to be formed in a number of ways.

C.7.2 Ports

Input two wire interface:

- 10 •in_valid
 - •in accept
 - •in data[7:0]
 - •in lastpel
 - •in lastline
- 15 Output two wire interface:
 - •out valid
 - •out_accept
 - •out_data[9:0]
 - •out last
- 20 mode(2:0)

nupdata[7:0], upaddr, upsel[3:0], uprstr, upwstr

ramtest

tdin, tdout, tph0, tckm, tcks ph0, ph1, notrst0

25 C.7.3 Mode

30

As selected by the input bus mode[2:0].

Mode register values 1 and 7 are not used.

In each of the above modes, the output pixels are represented as 10-bit values, not as bytes. No rounding or truncation takes place in this block. Where necessary, values are shifted left to use the same range.

C.7.3.1 Mode O:Fifo

The block simply acts as a FIFO store. The number of output pixels is exactly the same as at the input. The values are shifted left by two.

C.7.3.2 Mode 2: Repeat

Every line in the input scan is repeated to produce an output scan twice as high. Again, the pixel values are shifted left by two.

A-> ABACBDBCCDD

5

10

15

20

c.7.3.3 Mode 4: Lower

Each input line produces two output lines. In this "lower" mode, the second of these two lines (the lower on the display) is the same as the input line. The first of the pair is the average of the current input line and the previous input line. In the case of the first input line, where there is no previous line to use, the input line is repeated.

This should be selected where chroma samples are co-sited with the lower luma samples.

 $A\rightarrow ABAC(A+B)/2DB(B+C)/2C(C+D)/2D$

C.7.3.4 Mode 5: Upper

Similar to the "lower" mode, but in this case the input line forms the upper of the output pair, and the lower is the average of adjacent input lines. The last output line is a repeat of the last input line.

This should be selected where chroma samples are co-sited with the upper luma samples.

 $A\rightarrow AB(A+B)/2CBD(B+C)/2C(C+D)/2DD$

25 C.7.3.5 Mode 6: Central

This "central" mode corresponds to the situation where chroma samples lie midway between luma samples. In order to co-site the output chroma pixels with the luma pixels, a weighted average is used to form the output lines.

30 A -> AB(3A+B)/4C(A+3B)/4D(3B+C)/4(B+3C)/4 (3C+D)/4(C+3D)/4D

C.7.4 How It Works

There are two linestores, imaginatively designated "a" and "b". In "FIFO" and "repeat" modes, only linestore "a" is used. Each store can accommodate a line of up to 512 pixels (vertical upsampling should be performed before any horizontal upsampling). There is no restriction on the



length of the line in "FIFO" mode.

The input signals in_lastpel and in_lastline are used to indicate the end of the input line and the end of the picture. In_lastpel, it should be high coincident with the last pixel of each line. In_lastline, it should be high coincident with the last pixel of the last line of the picture.

The output signal out_last is high coincident with the last pixel of each output line.

In "repeat" mode, each line is written into store "a".

The line is then read out twice. As it is read out for the second time, the next line may start to be written.

In "lower", "upper" and "central" modes, lines are written alternately into stores "a" and "b". The first line of a picture is always written into store "a". Two tiny state machines, one for each store, keep track of what is in each store and which output line is being formed. From these states are generated the read and write requests to the linestore RAMs, and the signals that determine when the next line may overwrite the present data.

A register (lastaddr) stores the write address when in_lastpel is high, thereby providing the length of the line for the formation of the output lines.

C.7.5 UPI

15

20

25 This block contains two 512 x 8 bit RAM arrays, which may be accessed via the microprocessor interface in the typical way. There are no registers with microprocessor access.

SECTION C.8 The Horizontal Up-Samplers

C.8.1 Overview

5

10

In the present invention, top-Level Registers contain three identical Horizontal Up-samplers, one for each color component. All three are controlled independently and, therefore, only one need be described here. From the user's point of view, the only difference is that each Horizontal Up-sampler is mapped into a different set of addresses in the memory map.

The Horizontal Up-sampler performs a combined replication and filtering operation. In all, there are four modes of operation:

Table C.7.1 Horizontal Up-sampler Modes

Mode	Function
0	Straight-through (no processing). The reset state.
1	No up-sampling, filter using a 3-tap FIR filter.
2	x2 up-sampling and filtering
3	x4 up-sampling and filtering

C.8.2 Using a Horizontal Up-Sampler

The address map for each Horizontal Up-sampler consists of 25 locations corresponding to 12 13-bit coefficient registers and one 2-bit mode register. The number written to the mode register determines the mode of operation, as outlined in Table C.7.1. Depending on the mode, some or all of the coefficient registers may be used. The equivalent FIR filter is illustrated below.

Depending on the mode of operation, the input, x_n , is held constant for one, two or four clock periods. The actual coefficients that are programmed for each mode are as follows:

25

15



Table C.7.2 Coefficients for Mode 1

Coeff	All clock periods
k0	c00 _
k1	c10
k2	c20

Table C.7.3 Coefficients for Mode 2

Coeff	1st clock period	2nd clock period
KO	c00	j c01
λt	c10	c11
k2	c20	c21

Table C.7.4 Coefficients for Mode 3

Caeff	1st clock penod	2nd clock penod	3rd clock penod	4th clocx period
kO	c00	c01	c02	c03
k1	c10	c11	c12	c13
2	c20	c21	c22	c23

10

15

Coefficients which are not used in a particular mode need not be programmed when operating in that mode.

In order to achieve symmetrical filtering, the first and last pixels of each line are repeated prior to filtering. For example, when up-sampling by two, the first and last pixels of each line are replicated four times rather than two. Because residual data in the filter is discarded at the end of each line, the number of pixels output is still always exactly one, two or four times the number in the input stream.

Depending on the values of the coefficients, output samples can be placed either coincident with or shifted from the input samples. Following are some example values for coefficients in some sample modes. A "-" indicates that the value of the coefficient is "don't care." All values are in hexadecimal.

Table C.7.5 Sample Coefficients

	x2 up-sample, o/p pels	x2 up-sample, o/p pels in	x4 up-sample, c/b pe sin
Coefficient	coincident with Vp	between Vp	between vo
c00	0000	01BD	00E9
c01	0000	010B	0086
c02	•	•	012A
c03	•	•	0102
c10	0800	0538	0661
_ c11	0400 -	0538	0661
c12		-	0446
c13		•	029F
c20	0000	0108	0056
c21	0400	01BD	0069
c22	:	· ·	0290
c23	•		045F

C.S.3 Description of a Horizontal Up-Sampler

The datapath of the Horizontal Up-sampler is illustrated in Figure 168.

The operation is outlined below for the x4 upsample case. In addition, x2 upsampling and x1 filtering (modes 2 and 1) are degenerate cases of this, and bypass (mode 0) the entire filter, data passing straight from the input latch to the output latch via the final mux, as illustrated.

- 1) When valid data is latched in the input latch ("L"), it is held for 4 clock periods.
- 2) The coefficient registers (labelled "COEFF") are multiplexed onto the multipliers for one clock period, each in turn, at the same time as the two sets of four pipeline registers (labelled "PIPE") are clocked. Thus, for input data x_n, the first PIPE will fill up with the values coo.x_n, col.x_n, co2.x_n, co3.x_n.
- 3) Similarly, the second multiplier will multiply x_n by of its coefficients, in turn, and the third multiplier by all its coefficients, in turn.

It can be seen that the output will be of the form shown in Table C.7.6

Table C.7.6 Output Sequence for Mode 3

Clocki Penod	Output	
0	c20.x_ + c10.x_1 + c00.x_2	
1	c21.xn + c11.xn.1 + c01.xn.2	
2	c22.x ₁ + c12.x _{1.1} + c02.x _{1.2}	
3	c23.x ₁ + c13.x ₁₋₁ + c03.x ₁₋₂	

25 produces an individual pixel. Since each output pixel is dependent on the weighted values of 12 input pixels (although there are only three different values), this can

20

5

10

20



be thought of as implementing a 12 tap filter on x4 upsampled input pixels.

For x2 upsampling, the operation is essentially the same, except the input data is only held for two clock periods. Furthermore, only two coefficients are used and the "PIPE" blocks are shortened by means of the multiplexers illustrated. For x1 filtering, the input is only held for one clock period. As expected, one coefficient and one "PIPE" stage are used.

We now discuss a few notes about some peculiarities of the implementation in the present invention.

- 1) The datapath width and coefficient width (13 bit 2's complement) were chosen so that the same multiplier could be used, as was designed for the Color-Space Converter. These widths are more than adequate for the purpose of the Horizontal Up-sampler.
- 2) The multiplexers which multiplex the coefficients onto the multipliers are shared with the UPI readback. This has led to some complications in the structure of the schematics (primarily because of difficulty in CCODE generation), but the actual circuit is smaller.
- 3) As in the Color-Space Converter, carry-save multipliers are used, the result only being resolved at the end.

25 Control for the entire Horizontal Up-sampler can be regarded as a single two-wire interface stage which may produce two or four times the amount of data at its output as there is on its input. The mode which is programmed in via the UPI determines the length of a programmable shift register (bob). The selected mode produces an output pulse 30 every clock period, every two clock periods or every four This, in turn, controls the main state clock periods. machine, whose state is also determined by in valid, out_accept (for the two-wire interface) and the signal This signal is passed on from the vertical up-35 sampler and is high for the last pixel of every line. This allows the first and last pixels of each line to be

In the first term of the first

replicated twice-over and the clearing down of the pipeline between lines (the pipeline contains partially-processed redundant data immediately after a line has been completed).



C.9.1 Overview

5

10

The Color-Space Converter in the present invention (CSC) performs a 3x3 matrix multiplication on the incoming 9-bit data, followed by an addition:

$$\begin{bmatrix} y0 \\ y1 \\ v2 \end{bmatrix} = \begin{bmatrix} c01 & c02 & c03 \\ c11 & c12 & c13 \\ c12 & c22 & c23 \end{bmatrix} \times \begin{bmatrix} x0 \\ x1 \\ x2 \end{bmatrix} + \begin{bmatrix} c04 \\ c14 \\ c24 \end{bmatrix}$$

Where x0-2 are the input data, y0-2 are the output data and cnm are the coefficients. The slightly unconventional naming of the matrix coefficients is deliberate, since the names correspond to signal names in the schematics.

The CSC is capable of performing conversions between a number of different color spaces although a limited set of these conversions are used in Top-Level Registers. The design color-space conversions are as follows:

$$E_R,\,E_G,\,E_B\to Y,\,C_R,\,C_B$$

$$R, G, B \rightarrow Y, C_R, C_B$$

$$Y,\,C_R,\,C_B\to E_R,\,E_G,\,E_B$$

$$Y,\,C_R,\,C_B\to R,\,G,\,B$$

· Where R, G and B are in the range (0..511) and all other

1

quantities are in the range of (32..470). Since the input to the **Top-Level Registers** CSC is Y, C_R , C_B , only the third and fourth of these equations are of relevance.

In the CSC design, the precision of the coefficients was chosen so that, for 9 bit data, all output values were within plus or minus 1 bit of the values produced by a full floating point simulation of the algorithm (this is the best accuracy that it is possible to achieve). This gave 13 bit twos-complement coefficients for cx0-cx3 and 14 bit twos-complement coefficients for cx4. The coefficients for all the design conversions are given below in both decimal and hex.

Table	C 8 1	Coeffic	ante for	Various	Conversions
rabre	U.O.I	COSTITU.	renta for	VALIUUS	CONVELSIONS

	Ε ₄ ->Υ		A-:	A->Y		Eq	Y->	Y->#	
Coeff	Dec	Hex	Oec	Hex	Dec	Hex	Dec	-ex	
c 01	0.299	0132	0.256		1.0	0400	1,159	1 3443	
c02	0.587	0259	0.502		1,402	059C	1.539	CSSE	
c03	0.114	0075	0.098		0.0	0000	00	; 5050	
c Ω4	0.0	0000	16		-179.456	F4C8	-223 473	: =133	
c!1	. 0.5	0200	0.428		1.0	0400	1.169	04AD	
c12	-0.419	FE53	-0.358		-0.714	F025	-0.335	FCAB	
c:3	-0.081	FFAD	-0.070		-0.344	FEAO	-0.402	FE54	
c!4	128.0	0800	128		135.5	0878	139.7	AESC	
c21	-0.169	FF53	-0.144		1.0	0400	1.159	0440	
c22	-0.331	FEAD	-0.283		0.0	0000	0.0	0000	
c23	0.5	0200	0.427		1.772	0717	2.071	i 0849	
c24	128	0800	128		-225.816	F102	-283 34	EE42	

All these numbers are calculated from the fundamental equation:

$$Y = 0.299E_R + 0.587E_G + 0.0114E_B \label{eq:Y}$$
 and the following color-difference equations:

$$C_R = E_R - Y$$

$$C_B = E_B - Y$$

20

30

35



The equations in R, G and B are derived from these after the full-scale ranges of these quantities are considered. C.9.2 Using the Color-Space Converter

On reset, cO1, cl2, and c23 are set to 1 and all other coefficients are set to 0. Thus, y0=x0, y1=x1 and y2=x2 and all data is passed through unaltered. To select a color-space conversion, simply write the appropriate coefficients (from Table C.8.1, for example) into the locations specified in the address map.

Referring to the schematics, x0..2 correspond to in_data0..2 and y0..2 correspond to out_data0..2. Users should remember that input data to the CSC must be upsampled to 4:4:4. If this is not the case, not only will the color-space transforms have no meaning, but the chip will lock.

It should be noted that each output can be formed from any allowed combination of coefficients and inputs plus (or minus) a constant. Thus, for any given color-space conversion, the order of the outputs can be changed by swapping the rows in the transform matrix (i.e., the addresses into which the coefficients are written).

The CSC is guaranteed to work for all the transforms in Table C.8.1. If other transforms are used the user must remember the following:

- 25 1) The hardware will not work if any intermediate result in the calculation requires greater than 10 bits of precision (excluding the sign bit).
 - 2) The output of the CSC is saturated to 0 and 511. That is, any number less than 0 is replaced with 0 and any number more than 511 is replaced with 511. The implementation of the saturation logic assumes that the results will only be slightly above 511 or slightly below 0. If the CSC is programmed incorrectly, then a common symptom will be that the output appears to saturate all (or most of) the time.

25

30

35

C.9.3 Description of the CSC

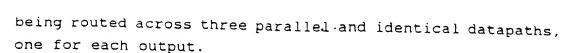
The structure of the CSC is illustrated in Figure 169, where only two of the three "components" have been shown because of space limitations. In the Figure, "register" or "R" implies a master-slave register and "latch" or "L" implies a transparent latch.

All coefficients are loaded into read-write UPI registers which are not shown explicitly in the Figure. To understand the operation, consider the following sequence with reference to the left-most "component" (that which produces output out data0):

- 1) Data arrives at inputs x0-2 (in_data0-2). This represents a single pixel in the input color-space. This is latched.
- 2)x0 is multiplied by c01 and latched into the first pipeline register. x1 and x2 move on one register.
 - 3) x1 is multiplied by c02, added to (x1.c01) and latched into the next pipeline register. x2 moves on one register.
- 4) x2 is multiplied by c03 and added to the result of (3), producing (x1.c01 + x2.c02 + x3.c03). The result is latched into the next pipeline register.
 - 5) The result of (4) is added to c04. Since data is kept in carry-save format through the multipliers, this adder is also used to resolve the data from the multiplier chain. The result is latched in the next pipeline register.
 - 6) The final operation is to saturate the data. Partial results are passed from the resolving adder to the saturate block to achieve this.

It can be seen that the result is y0, as specified in the matrix equation at the start of this section. Similarly, y1 and y2 are formed in the same manner.

Three multipliers are used, with the coefficients as the multiplicand and the data as the multiplicator. This allows an efficient layout to be achieved, with partial results flowing down the datapath and the same input data



To achieve the reset state described in Section C.9.2, each of the three "components" must be reset in a different way. In order to avoid having three sets of schematics and three slightly different layouts, this is achieved by having inputs to the UPI registers which are tied high or low at the top level.

The CSC has almost no control associated with it.

Nevertheless, each pipeline stage is a two-wire interface stage, so there is a chain of valid and accept latches with their associated control (in_accept = out_accept_r + lin_valid_r). The CSC is, therefore, a 5-stage deep two-wire interface, capable of holding 10 levels of data when stalled.

The output of the CSC contain re-synchronizing latches because the next function in the output pipe runs off a different clock generator.

SECTION C.10 Output Controller

C.10.1 Introduction

The output controller, in accordance with the present invention, handles the following functions:

- •It provides data in one of three modes
 - 24-bit 4:4:4
 - 16-bit 4:2:2
 - 8-bit 4:2:2
- •It aligns the data to the video display window defined by the vsync and hsync pulses and by programmed timing registers
 - •It adds a border around the video window, if required

C.10.2 Ports

5

- 15 Input two wire interface:
 - •in_valid
 - •in_accept
 - •in data[23:0]

Output two wire interface:

- 20 •out valid
 - •out_accept
 - •out data[23:0]
 - •out active
 - out window
- 25 •out_comp[1:0]

in vsync, in hsync

nupdata[7:0], upaddr[4:0], upsel, rstr, wstr
tdin, tdout, tphO, tckm, tcks chiptest
phO, phl, notrstO, notrst1

30 C.10.3 Out Modes

The format of the output is selected by writing to the opmode register.

C.10.3.1 Mode O

This mode is 24-bit 4:4:4 RGB or YCrCb. Input data passes directly to the output.

C.10.3.2 Modes 1 and 2

These modes present 4:2:2 YCrCb. Assuming in_data[23:16] is Y, in_data[15:8] is Cr and in_data[7:0] is Cb.

C.10.3.2.1 Mode 1

In 16-bit YCrCb, Y is presented on out_data[15:8]. Cr and Cb are time multiplexed on out_data[7:0], Cb first. Out_data[23:16] is not used.

C.10.3.2.2 Mode 2

In 8-bit YCrCb, Y,Cr and Cb are time multiplexed on out_data[7:0] in the order Cb, Y, Cr, Y. Out_data[23:8] is not used.

C.10.3.3 Output Timing

The following registers are used to place the data in a video display window.

- •vdelay The number of hsync pulses following a vsync pulse before the first line of video or border.
 - •hdelay The number of clock cycles between hsync and the first pixel of video or border.
 - •height The height of the video window, in lines.
- •width The width of the video window, in pixels.
 - •north, south The height of the border, respectively, above and below the video window, in lines.
 - •west, east The width of the border, respectively, to the left and to the right of the video window, in pels.
- The minimum vdelay is zero. The first has is the first active line. The minimum value that can be programmed into hdelay is 2. Note, however, that the actual delay from in_has to the first active output pixel is hdelay+1 cycles.
- Any edge of the border can have the value zero. The color of the border is selected by writing to the registers border_r, border_g and border_b. The color of the area outside the border is selected by writing to the registers blank_r, blank_g and blank_b. Note that the multiplexing
- performed in output modes 1 and 2 will also affect the border and blank.components. That is, the values in these registers correspond with in data[23:16], in_data[15:8] and

in_data [7:0].

C.10.4 Output Flags

•out_active indicates that the output data is part of the active window, i.e., video data or border.

•out_window indicates that the output data is part of the video window.

•out_comp[1:0] indicates which color component is present on out_data[7:0] in output modes 1 and 2. In mode 1, 0=Cb, 1=Cr. In mode 2, 0=Y, 1=Cr, 2=Cb.

10 C.10.5 Two-Wire Mode

The two-wire mode of the present invention is selected by writing 1 to the two wire register. It is not selected following reset. In two wire mode, the output timing registers and sync signals are ignored and the flow of data through the block is controlled by out_accept. Note that in normal operation, out_accept should be tied high.

C.10.6 Snooper

15

25

There is a super-snooper on the output of the block which includes access to the output flags.

20 C.10.7 How It Works

Two identical down-counters keep track of the current position in the display. "Vcount" decrements on haynes and loads from the appropriate timing register on vsync or at its terminal count. "Hount" decrements on every pixel and loads on hayne or at its terminal count. Note that in output mode 2, one pixel corresponds to two clock cycles.



C.11.1 Overview

5

10

15

20

30

35

Top-Level Registers in the present invention contain two identical Clock Dividers, one to generate a PICTURE_CLK and one to generate an AUDIO_CLK. The Clock Dividers are identical and are controlled independently. Therefore, only one need be described here. From the user's point of view, the only difference is that each Clock Divider's divisor register is mapped into a different set of addresses in the memory map.

The Clock Divider's function is to provide a 4X sysclk divided clock frequency, with no requirement for an even mark-space ratio.

The divisor is required to lay in the range ~0 to ~16,000,000 and, therefore, it can be represented using 24bits with the restriction that the minimum divisor be 16. This is because the Clock Divider will approximate an equal mark-space ratio (to within one sysclk cycle) by using divisor/2. As the maximum clock frequency available is sysclk, the maximum divided frequency available is sysclk/2. Furthermore, because four counters are used in cascade divisor/2 must never be less than 8, else the divided clock output will be driven to the positive power rail.

25 C.11.2 Using a Clock Divider

The address map for each Clock Divider consists of 4 locations corresponding to three 8-bit divisor registers and one 1-bit access register. The Clock Divider will power-up inactive and is activated by the completion of an access to its divisor register.

The divisor registers may be written in any order according to the address map in Table C.10.1. The Clock Divider is activated by sensing a synchronized 0 to 1 transition in its access bit. The first time a transition is sensed, the Clock Divider will come out of reset and generate a divided clock. Subsequent transitions (assuming

the divisor has also been altered) will merely cause the Clock Divider to lock to its new frequency "on-the-fly." Once activated, there is no way of halting the Clock Divider other than by Chip RESET.

Table C.10.1 Clock Divider Registers

Address	Register
006	access bit
015	divisor MSB
10b	divisor
116	divisor LSB

Any divisor value in the range 16 to 16,777,216 may be used.

C.11.3 Description of the Clock Divider

The Clock Divider is implemented as four 22 bit counters which are cascaded such that as one counter carries, it will activate the next counter in turn. A counter will count down the value of divisor/4 before carrying and, therefore, each counter will take it, in turn, to generate a pulse of the divided clock frequency.

After carrying, the counter will reload with divisor/8 and this is counted down to produce the approximate equal mark-space ratio divided clock. As each counter reloads from the divisor register when it is activated by the previous counter, this enables the divided clock frequency to be changed on the fly by simply altering the contents of the divisor.

Each counter is clocked by its own independent clock generator in order to control clock skew between counters precisely and to allow each counter to be clocked by a different set of clocks.

A state machine controls the generation of the divisor/4 and divisor/8 values and also multiplexes the correct source clocks from the PLL to the clock generators. The

20

25

15

10

10

counters are clocked by different clocks dependent on the value of the divisor. This is because different divisor values will produce a divided clock whose edges are placed using different combinations of the clocks provided from the PLL.

C.11.4 Testing the Clock Divider

The Clock Divider may be tested by powering up the Chip with CHIPTEST High. This will have the effect of forcing all of the clocked logic in the Clock Divider to be clocked by sysclk, as opposed to, the clocks generated by the PLL.

The Clock Divider has been designed with full scan and, thus, may subsequently be tested using standard JTAG access, as long as the Chip has been powered up as above.

The functionality of the Clock Divider is NOT guaranteed if CHIPTEST is held High while the device is running in normal operation.



SECTION C.12 Address Maps

C.12.1 Top Level Address Map

Notes:-

5

1) The register for the Top Level Address Map as set forth in Table C.11.1 are the names used during the design. They are not necessarily the names that will appear on the datasheet.

2) Since this is a full address map, many of the locations listed here include locations for test only.

REGISTER NAME	Address	Bits	COMMENT
BU_EVENT	0x0	8	Write 1 to reset
BU_MASK	0x1	8	R/W
BU_EN_INTERRUPTS	0×2	1	R/W
BU_WADDR_COD_STD	0x4	2	R/W
BU_WACDR_ACCESS	0x5	1	RW- access
SU_WADDR_CTL1	0x6	3	R/W
OWOR_TZAJ_NI_ZBNIJ_RODASZIC_UE	0x8	3	RW
BU_DISPADDR_LINES_IN_LAST_ROW1	0x9	3	R/W
SWOR_TRANSPORTED SWORT STORY	0xa	3	R/W
BU_DISPADDR_ACCESS	Oxb	1	R/W- access
BU_DISPADDR_CTL0	0xc	8	R/W
BU_DISPADDR_CTL1	0xd	1	RW
BU_BM_ACCESS	0x10	1	R/W- access
BU_BM_CTL0	0x11	2	RW
BU_BM_TARGET_IX	0x12	4	R/W
BU_BM_PRES_NUM	0x13	8	R/W-asynchronous
BU_BM_THIS_PNUM	0x14	8	RW
SU_SM_PIC_NUMO -	0x15	8	RW
SU_SM_PIC_NUM1	0x16	8	P/W
BU_BM_PIC_NUM2	0x17	8	P/W
SU_SM_TEMP_REF	0x18	5	RO

Table C.11.1 Top-Level Registers A Top Level Address Map

- HEGISTER NAME	Add	Bits	COMMENT
BU_ADDRGEN_KEYHOLE_ADDR_MSB	0×28	1	R/W- Address generator
BU_ADDRGEN_KEYHOLE_ADDR_1SB	0x29	8	keyhole See
BU_ADDRGEN_KEYHOLE_DATA	0x2a	8	Table C.:! 2 for contents
BU_IT_PAGE_START	0x30	5	RVW
BU_IT_READ_CYCLE	0x31	4	RW
BU_IT_WRITE_CYCLE	0x32	4	PW
SU_IT_REFRESH_CYCLE	0x33	4	F/W
SU_IT_RAS_FALLING	0x34	1 4	PAV
SU_IT_CAS_FALLING	0x35	4	PAV
BU_IT_CONFIG	0x35	1	PW
9U_OC_ACCESS	0x40	1	RW- access .
BU_OC_MODE	0x41	2	RW
BU_OC_2WIRE	0x42	1	RW
BU_OC_BORDER_R	0x49	8	RW
BU_OC_BORDER_G	0x4a	8	P/W
BU_OC_BOADER_B	0x4b	8	RW
BU_OC_BLANK_R	0x4d	8	RW
BU_OC_BLANK_G	0x4e	8	RW
BU_OC_BLANK_B	0x4f	8	RW .
BU_OC_HDELAY_1	0x50	3	R/W i
BU_OC_HOELAY_0	0x51	8	RW
BU_OC_WEST_1	0x52	3	PVW
BU_OC_WEST_0	0x53	8	P/W
BU_OC_EAST_1	0x54	3	FVW
BU_OC_EAST_0	0x55	8	R/W
BU_OC_WIDTH_1	0×56	3	R/W
BU_OC_WIDTH_0	0x57	8	RW
BU_OC_VDELAY_1	0x58	3	P/W
SU_OC_VDELAY_0	0x59	8	R/W
SU_OC_NORTH_1	0x5a	3	RW
BU_OC_NORTH_0	0x5b	8	R/W :
BU_OC_SOUTH_1	0x5c	3	R/W
BU_OC_SOUTH_0	0x5d	8	P/W
BU_OC_HEIGHT_1	0x5e	3	R/W ,
BU_OC_HEIGHT_0	0x51	8	R/W

Table .C.11.1 Top-Level Registers A Top
Level Address Map (contd)



	252:2				T	
	REGISTER NAME	- Addre	ss	Bits	COMMENT	
	BU_IF_CONFIGURE	0x60		5	RVW	==
	BU_UV_MODE	0x61		6	RW- xonnxonn	
	BU_COEFF_KEYADOR	0x62		7	FW - See Table C.:: 3	
	BU_COEFF_KEYDATA	0x63		8	for contents.	
1	BU_GA_ACCESS	0x58	T	1	P/W	
	BU_GA_BYPASS	0x69	Ť	1	RW	
į	BU_GA_RAMO_ADDR	0x6a	T	8	F/W	
	BU_GA_RAMO_DATA	0x6b	十	8	RW	_
	BU_GA_RAM1_ADDR	0x6c	$\overline{}$	8	RW	
	BU_GA_RAM1_DATA	0x6d	$\neg +$	8	RW	
	SU_GA_RAM2_ADDR	0x6e	-	3	RW	
	BU_GA_RAM2_DATA	0x61		3	RW	
Ī	BU_OIVA_3	0x70	1		RW	
	BU_DIVA_2	0x71	- -		PW	
	BU_DIVA_1	0x72	- 8		RW	
Ī	BU_DIVA_0	0x73	8		RW	
Γ	BU_DIVP_3	0x74	1		F/W	
Γ	BU_DIVP_2	0x75	a		PW	_
Γ	SU_DIVP_1	0x75	8		P/W	
	9U_DIVP_0	0x77	8	-		_; ;
8	BU_PAD_CONFIG_1	0x78	7		₹W	_;
	BU_PAD_CONFIG_0	0x79	8		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	<u>.</u>
5	BU_PLL_RESISTORS	0x7a	18		w	- :
E	BU_REF_INTERVAL	0x7b	8		w	ij
Ε	BU_REVISION	0xff	8		IO- revision	-;
ī	he following registers are in the "test space".		1		IC- TEVISION	j
_	hey are unlikely to appear on the datasheet	 				1
_	U_BM_PRES_FLAG	0x80	1	R	w	!
_	U_SM_EXP_TR	0x81	"	_ n	nese registers are	 ;
_	U_BM_TR_DELTA	0x82	••	m	issing on reva	į
	U_8M_ARR_IX	0x83	2	PV	w	-
	U_9M_OSP_IX	0x84	2	P/	w	•
31	U_BM_ADY_IX	0×85	2	PV	w	•
91	J_BM_BSTATE3	0×86	2	P/	w	•
81	J_BM_BSTATE2	0x87	2	P.	w	•

Table C.11.1 Top-Level Registers A Top Level Address Map (contd)

REGISTER NAME	656	-		
	Addre	ss	3:15	COMMENT
BU_BM_BSTATE!	0x88		2	RW
BU BM CTITE	0x89	i	2	AW
BU_BM_STATE	0x8a	1	5	₽W
BU_BM_FROMPS	0x85	İ	1	P.W
BU_BM_FROMFL	0x8c		1 !	AW
BU_CA_COMP0_SNP3	0x90	i	3	RW - These are the three
BU_DA_COMRO_SNP2	0x91	! !	3	snoopers on the display
SU_DA_COMPO_SNP1	0x92	1 8	3	
BU_CA_COMPO_SNPO	0x93	8		address generators
BU_CA_COMP1_SNP3	Cx94	a		address output
BU_DA_CCMP1_SNP2	0x95	8		
BU_DA_COMP1_SNP1	0×96	8	\neg	
BU_DA_COMP1_SNP0	0x97	8	\neg	
BU_DA_CCMP2_SNP3	0x98	8	\exists	
EU_CA_COMP2_SNP2	0x99	5	\neg	
BU_CA_COMP2_SNP1	0x9a	8		,
SU_DA_CCMP2_SNP0	0x95	8	$\overline{}$	
1_RDDA_A1MAF_VU_UE	0xa0	8	F	W - upi test access into
BU_UV_RAM1A_ADDR_0	0xa1	8		:
BU_UV_RAM1A_DATA	0xa2	8	- 1	ie verdcal ussamplers.
SU_UV_RAM18_AODR_1	0xa4	18	- R	AMS
BU_UV_RAM1B_ADDR_0	0xa5	8	7	į
SU_UV_RAM1B_DATA	0xa6	8	7	
BU_UV_RAM2A_ADDR_1	0xa8	8	7	
BU_UV_RAM2A_ADDR_0	0xa9	а		
ATAG_ASMAP_VU_UE	Oxaa	8	7	1
BU_UV_RAM2B_ADDR_1	Oxac	8	1	
BU_UV_RAM29_ADDR_0	0xad	8		i :
BU_UV_RAM2B_DATA	Oxae	8	7	i
EU_WA_ADDR_SNP2	0xb0	8	RW	V - snooper on the write
BU_WA_ACOR_SNP1	0x51	в	7	!
BU_WA_ADDR_SNPO	0xb2	8	م م	ress generator address
BU_WA_DATA_SNP1	0xb4	3	FM	/ - snooper on data
BU_WA_DATA_SNP0	0xb5	a	7	or WA
Table C 11 1 Top-			1	

Table C.11.1 Top-Level Registers A Top Level Address Map (contd)

: SECIETTO MANAGE		T -	1
REGISTER NAME -	Address	Sits	COMMENT
BU_IF_SNPO_1	CxD8	8	PVW - Three shoopers on
3U_!F_SNP0_0	0x59	8	the dramif data outputs
BU_IF_SNP1_1	0x5a	8	and distance and d
BU_IF_SNP1_0	0x00	8	
BU_IF_SNP2_1	0xbc	8	
BU_IF_SNP2_0	0xbd	8	
BU_IFRAM_ADDR_1	0xc0	1	RW - upi access it IF RAM -
BU_IFRAM_ADDR_0	0xc1	8	
BU_IFRAM_DATA	0xc2	8	
BU_OC_SNP_3	0xc4	8	R/W - snooper on output of
BU_OC_SNP_2	0xc5	8	ן מוחס
BU_CC_SNP_1	0xc6	8	
BU_CC_SNP_0	0xc7	8	;
BU_YAPLL_CONFIG	0xc8	8	R/W
BU_BM_FRONT_BYPASS	Охса	1	RW

Table C.11.1 Top-Level Registers A Top Level Address Map (contd)

C.12.1 Address Generator Keyhole Space

Notes on address generator keyhole table:

- 1) All registers in the address generator keyhole take up 4 bytes of address space regardless of their width. The missing addresses (0x00, 0x04 etc.) will always read back zero.
- 2) The access bit of the relevant block (dispaddr or waddrgen) must be set before accessing this keyhole.

10



Keyhole Pegister Name Address Bits Comments BU_OISPACOR_BUFFERO_BASE_MSB 0x01 2 18 oit BU_OISPACOR_BUFFERO_BASE_MID 0x02 3 register BU_OISPACOR_BUFFERO_BASE_LSB 0x03 3 Must be BU_OISPACOR_BUFFERO_BASE_LSB 0x05 2 Must be BU_OISPACOR_BUFFERI_BASE_MSB 0x05 2 Must be BU_OISPACOR_BUFFERI_BASE_LSB 0x07 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x09 2 Must be BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 8 Loaded BU_OISPACOR_BUFFERI_BASE_LSB 0x00 <t< th=""><th></th><th></th><th></th><th></th></t<>				
Accress Su_DISPACOR_BUFFERO_BASE_MSB 0x01 2 18 oit	Keyhole Register Name	Keynole		Comments
SU_DISPACOR_BUFFERO_BASE_ISB		Address	:	:
BU_DISPACOR_BUFFERO_BASE_LSB	BU_DISPADDR_BUFFERO_BASE_MSB	0x01	. 2	18 cit
Must be	BU_DISPADDR_BUFFERO_BASE_MID	3x02	; 3	register -
	BU_DISPACOR_BUFFERO_BASE_1SB	Gx03	¦ 3	Mustbe
BU_DISPADDR_BUFFER1_BASE_MSB				
BU_DISPADDR_BUFFER1_BASE_MID 0x06 8 Loaded	AN DISPADOR RUSSERY PACE MOR	1	1 -	1
SU_DISPADDR_BUFFER1_BASE_LSB	<u></u>	0x05	. 2	Must be
SU_DISPADDR_BUFFER2_BASE_MSB Ox09 2		0x06	8	Loaded
BU_DISPADDR_BUFFER2_BASE_MID Ox0a 8 BU_DISPADDR_BUFFER2_BASE_LSB Ox0b 3 BU_DISPADDR_BUFFER2_BASE_LSB Ox0d 2 Test only	BU_DISPADDR_BUFFER1_BASE_LSB	0×07	8	<u> </u>
SU_DISPACDR_BUFFER2_BASE_LSB	SU_DISPADOR_BUFFER2_BASE_MS8	0x09	2	Must be
BU_DLDPATH_LINEO_MID	BU_DISPADDR_BUFFER2_BASE_MID	50x0	8	Loaded
BU_DLDPATH_LINEO_MID	BU_DISPACOR_BUFFER2_BASE_LSB	Cx0b	3	
BU_DLDPATH_LINEO_LSB 0x01 8 BU_DLDPATH_LINE1_MSB 0x11 2 Test only BU_DLDPATH_LINE1_MID 0x12 3 0x13 8 BU_DLDPATH_LINE1_LSB 0x13 8 0x15 2 Test only BU_DLDPATH_LINE2_MSB 0x15 2 Test only BU_DLDPATH_LINE2_LSB 0x17 8 0x17 8 BU_DLDPATH_VBCNTO_MSB 0x19 2 Test only BU_DLDPATH_VBCNTO_MID 0x1a 3 0x1b 3 BU_DLDPATH_VBCNTO_LSB 0x1b 3 0x1d 1 2 Test only BU_DLDPATH_VBCNT1_MSB 0x1d 1 2 Test only 0x1d 1 3 0x1d 1 2 Test only BU_DLDPATH_VBCNT1_MSB 0x1d 1 2 Test only 0x1d 1 3 0x1d 1 2 Test only 0x1d 1 0x1d 1 0x1d 1 0x1d 1 0x1d 1 0x1d	BU_DLOPATH_LINEO_MSB	0x0d	2	Test only
BU_DLDPATH_LINE1_MID 0x12 3 BU_DLDPATH_LINE1_MID 0x12 3 BU_DLDPATH_LINE1_LSB 0x13 8 BU_DLDPATH_LINE2_MSB 0x15 2 Test only BU_DLDPATH_LINE2_MID 0x16 8 BU_DLDPATH_LINE2_LSB 0x17 8 BU_DLDPATH_VBCNT0_MSB 0x19 2 Test only BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1d 3 BU_DLDPATH_VBCNT1_MSB 0x1d 2 Test only BU_DLDPATH_VBCNT1_LSB 0x1f 3 BU_DLDPATH_VBCNT1_LSB 0x1f 3 BU_DLDPATH_VBCNT1_LSB 0x1f 3 BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only	BU_DLDPATH_LINEO_MID	0x0e	а	
BU_DLDPATH_LINE1_LSB 0x12 3 BU_DLDPATH_LINE2_MSB 0x15 2 Test only BU_DLDPATH_LINE2_MID 0x16 8 BU_DLDPATH_LINE2_LSB 0x17 8 BU_DLDPATH_VBCNT0_MSB 0x19 2 Test only BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1 BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1l 3 BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only	BU_DLDPATH_LINEO_LSB	Ox01	8	
BU_DLDPATH_LINE1_LSB	BU_DLDPATH_LINE1_MSB	0x11	2	Test only
BU_DLDPATH_LINE2_MSB 0x15 2 Test only BU_DLDPATH_LINE2_MID 0x16 8 BU_DLDPATH_LINE2_LSB 0x17 8 BU_DLDPATH_VBCNT0_MSB 0x19 2 BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1 BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1l 3 BU_DLDPATH_VBCNT1_LSB 0x1l 3 BU_DLDPATH_VBCNT2_MSB 0x2l 2 Test only BU_DLDPATH_VBCNT2_MSB 0x2l 3 Test only	BU_DLDPATH_LINE1_MID	0x12	9	
BU_DLDPATH_LINE2_MID 0x16 8 BU_DLDPATH_LINE2_LSB 0x17 8 BU_DLDPATH_VBCNT0_MSB 0x19 2 Test only BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1 BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1l 3 BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only	BU_DLDPATH_LINEI_LSB	0x13	8	
BU_DLDPATH_LINE2_LSB 0x17 8 BU_DLDPATH_VBCNT0_MSB 0x19 2 Test only BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1 BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1f 3 BU_DLDPATH_VBCNT1_LSB 0x2f 3 BU_DLDPATH_VBCNT2_MSB 0x2f 2 Test only BU_DLDPATH_VBCNT2_MID 0x2z 3	BU_DLDPATH_LINE2_MSB	0x15	2	Test only
BU_DLDPATH_VBCNT0_MSB 0x19 2 Test only BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1 BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1f 1 BU_DLDPATH_VBCNT2_MSB 0x21 2 BU_DLDPATH_VBCNT2_MID 0x2e 3	BU_DLDPATH_LINE2_MID	0x16	8	
BU_DLDPATH_VBCNT0_MID 0x1a 3 BU_DLDPATH_VBCNT0_LSB 0x1b 3 BU_DLDPATH_VBCNT1_MSB 0x1d 1.2 Test only BU_DLDPATH_VBCNT1_MID 0x1e 1.3 BU_DLDPATH_VBCNT1_LSB 0x1l 1.3 BU_DLDPATH_VBCNT2_MSB 0x2l 2 Test only BU_DLDPATH_VBCNT2_MSB 0x2l 3 Test only	BU_DLOPATH_LINE2_LSB	0x17	8	
BU_DLDPATH_VBCNT0_LSB 0x1b i 3 BU_DLDPATH_VBCNT1_MSB 0x1d i 2 Test only BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1l 3 BU_DLDPATH_VBCNT2_MSB 0x2l 2 Test only BU_DLDPATH_VBCNT2_MID 0x2e 3 Image: Control only	BU_DLDPATH_VBCNTO_MSB	Cx19	2	Test only
BU_DLDPATH_VBCNT1_MSB 0x1d 1 2 Test only BU_DLDPATH_VBCNT1_MID 0x1e 3 : BU_DLDPATH_VBCNT1_LSB 0x1l 3 : BU_DLDPATH_VBCNT2_MSB 0x2l 2 Test only BU_DLDPATH_VBCNT2_MID 0x2e 3 :	SU_DLDPATH_VBCNTO_MID	Oxia	3	1
BU_DLDPATH_VBCNT1_MID 0x1e 3 BU_DLDPATH_VBCNT1_LSB 0x1f 3 BU_DLDPATH_VBCNT2_MSB 0x2f 2 Test only BU_DLDPATH_VBCNT2_MID 0x2e 3	BU_DLDPATH_VBCNTO_LSB	Oxib	3	
BU_DLDPATH_VBCNT1_LSB 0x11 3 BU_DLDPATH_VBCNT2_MSB 0x21 2 Test only BU_DLDPATH_VBCNT2_MID 0x22 3	BU_DLDPATH_VBCNT1_MS8	0x1d	2	Test only
BU_DLDPATH_VBCNT2_MS8 0x21 2 Test only BU_DLDPATH_VBCNT2_MID 0x22 3	SU_DLOPATH_VBCNT1_MID	Oxie	3	:
BU_DLDPATH_VBCNT2_MID 0x22 S	BU_DLDPATH_VBCNT1_LSB	11x0	3	
	BU_OLOPATH_VBCNT2_MS8	0x21	2	Test only
SU_DLDPATH_VBCNT2_LSB 0x23	BU_DLDPATH_VBCNT2_MID	0x22	3	
	BU_DLDPATH_VBCNT2_LSB	0x23	3 !	



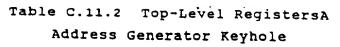
	Keliote	3	
i ; ; Keynole Register Name	Keynole		
l l	Address	i 3its	Comments
BU_DISPACOF_CCMP0_OFFSET_MSB	1 Cx25	2	: Musibe
BU_DISPADOR_COMPO_OFFSET_MID	0×25	3	Loaces
BU_DISPADOR_CCMP0_OFFSET_LSB	0:27	8	
BU_DISPADOR_COMP1_OFFSET_MSB	0x29	2	Must be
BU_DISPACER_COMP1_OFFSET_MID	0x2a	3	Loaced
BU_DISPACCH_COMP!_OFFSET_LSB	0x25	8	
BU_DISPADOR_CCMPZ_OFFSET_MSB	0x2d	2	Must be
BU_DISPADDR_CCMP2_OFFSET_MID	0×2e	8	Loaded
BU_DISPADOR_COMP2_OFFSET_LSB	0x21	8	
BU_DISPADDR_COMPO_VBS_MSB	0x31	2	Must be
BU_DISPACOR_COMPO_VBS_MID	0x32	8	Loacec
BU_DISPADDR_COMPO_VBS_LSB	0x33	3	
BU_DISPADDR_CCMP1_VBS_MSB	0x35	2	Must be
BU_DISPADDR_COMP1_VBS_MID	0x36	8	Loaded
BU_DISPADDR_COMP1_VBS_LSB	0x37	8	,
BU_DISPADDR_COMP2_VBS_MSB	0x39	2	Must be
BU_DISPADDR_COMP2_VBS_MID	0x3a	8	Loaded
BU_DISPADDR_COMP2_VBS_LSB	0x3b	3	
BU_ADDR_COMPO_HBS_MSB	0x3d	2	Must be
BU_ADDR_COMPO_HBS_MID	0x3e	8	Loaded
BU_ADDR_COMPO_HBS_LSB	10x31	8	1
BU_ADDR_COMP1_HBS_MSB	0x41	2	Must be
BU_ADDR_COMP1_HBS_MID	0x42	3	Loaced
BU_ADDR_COMP1_HBS_LSB	0x43	8	!
SU_ACOR_COMP2_HBS_MSB	0x45	2	Must be
BU_ADDR_COMP2_HBS_MID	0x46	9	Loaded
BU_ADDR_COMP2_HBS_LSB	0x47	8	
BU_DISPADDR_COMP0_HBS_MSB	0x49	2	Must be :
BU_DISPADOR_COMPO_HBS_MID	0x4a	9	Loaced
BU_DISPACOR_COMPO_HBS_LSB	Cx4b	8	
BU_DISPACER_COMP1_HBS_MSB	0x4d	2	Must be
BU_DISPADDR_COMP!_HBS_MIO	0x4e	3	Loaded
BU_DISPADOR_COMP1_HBS_LSB	0x4!	8	



Kaubala 2 Na	Keynoie	_
Keyhole Pegister Name	Address	Bits Comments
BU_DISPADOR_COMP2_HBS_MSB	ı Çx51 1 2	Musi te
BU_DISPACOR_COMP2_HBS_MID	Cx52 ; 3	
BU_DISPADDR_COMP2_HBS_LSB	0x53 3	
BU_DISPADDR_CNT_LEFTO_MSB	0x55 2	Test only
BU_DISPADDR_CNT_LEFT0_MIO	0x56 3	 !
BU_DISPADDR_CNT_LEFT0_LSB	0x57 3	!
BU_DISPADER_CNT_LEFT1_MSB	0x59 2	Test only
BU_DISPADDR_CNT_LEFT1_MID	0x5a 3	
BU_DISPADDR_CNT_LEFT1_LSB	0x5b 8	
BU_DISPADOR_CNT_LEFT2_MSB	0x5d 2	Test only
BU_DISPADOR_CNT_LEFT2_MID	0x5e 3	
BU_DISPADOR_CNT_LEFT2_LSB	0x51 3	
BU_DISPADDR_PAGE_ADDR0_MSB	0x61 : 2	Test only
BU_CISPADDR_PAGE_ADDRO_MID	0x62 8	
BU_DISPADDR_PAGE_ADDRC_LSB	0x63 8	i
BU_DISPADDR_PAGE_ADDR1_MS3	0x65 2	Test only
BU_DISPADDR_PAGE_ADDR1_MID	0x66 3	
BU_DISPADDR_PAGE_ADDR1_LSB	0x67 8	1
BU_DISPADDR_PAGE_ADDR2_MSB	0x69 2	Test only
BU_DISPADDR_PAGE_ADDR2_MID	Ox6a S	
BU_DISPADDR_PAGE_ADDR2_LSB	0x6b 8	
BU_DISPADDR_BLOCK_ADDRO_MSB	0x6d 2	Test only
SU_DISPADDR_BLOCK_ADDRO_MID	0x5e 3	
BU_DISPADOR_BLOCK_ADDRO_LSB	0x61 ! 3	
BU_DISPADDR_BLOCK_ADDR1_MSB	0x71 2	Test only
BU_DISPADDR_BLOCK_ADDR1_MIO	0x72 8	
BU_DISPADDR_BLOCK_ADDR1_LSB	0x73 8	
BU_DISPADOR_BLOCK_ADDR2_MSB	0x75 2	Test triy
SU_DISPACOR_BLOCK_ADDR2_MID	0x76 3	
BU_DISPADDR_BLOCK_ADDR2_LSB	0x77 8	!
BU_DISPADDR_BLOCKS_LEFT0_MSB	0x79 2	¹ Testionly
BU_DISPADOR_BLOCKS_LEFTO_MID	0x7a 8	
BU_DISPADOR_BLOCKS_LEFTO_LS8	0x7b 9	



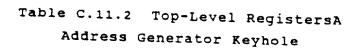
Keyhole Register Name	Keyhole Address	3:6	Comments
5U_DISPADDR_BLOCKS_LEFT1_MSB	0x7d	2	r Test only
BU_DISPADDR_BLOCKS_LEFT!_MID	0x7e	l a	1
BU_DISPADDR_BLOCKS_LEFT1_LSB	0x71	8	
BU_DISPADDR_BLOCKS_LEFT2_MSB	0x81	2	Test only
BU_DISPADDR_BLOCKS_LEFT2_MID	0x82	8	
BU_DISPADDR_BLOCKS_LEFT2_LSB	0x83	8	į
BU_WADDR_BUFFER0_BASE_MSB	0x85	2	Must be
BU_WACOR_BUFFERO_BASE_MID	0x86	8	Loaded
BU_WADDR_BUFFERO_BASE_LSB	0x87	8	
BU_WACOR_BUFFER1_BASE_MSB	0x89	2	Must be
BU_WADDR_BUFFER1_BASE_MID	0x8a	8	Loaded
BU_WADOR_BUFFER1_BASE_LSB	0x8b	8	
BU_WACOR_BUFFER2_BASE_MSB	0x8d	2	Musi be
BU_WACOR_SUFFER2_SASE_MID	0x8e	8	Loaded
BU_WADDR_BUFFER2_BASE_LSB	0x8f	8	
BU_WADDR_COMPO_HMBADDR_MSB	0x91	2	Test only
BU_WADDR_COMPO_HMBADDR_MID	0x92	8	
3U_WACDR_COMPO_HMBADDR_LSB	0x93	8	
SU_WADDR_COMP1_HMBADDR_MSB	0x95	2	Test only
SU_WADDR_COMP1_HMBADDR_MID	0x96	8	
BU_WADDR_COMP1_HMBADDR_LSB	0x97	8	
BU_WADDR_COMP2_HMBADDR_MSB	0x99	2	Test only
BU_WADDR_COMP2_HMBADDR_MID	0x9a	9	
BU_WADDR_COMP2_HMBADDR_LSB	0x9b	8	
BU_WADDR_COMPO_VMBADDR_MSB	0x9d	2	Test only
BU_WACDR_COMPO_VMBADOR_MID	0x9e	8	<u> </u>
BU_WADDR_COMPO_VMBADDR_LSB	0x9!	8	
SU_WADDR_COMP1_VMBADDR_MSB	0xa1	2	Test only
BU_WADDR_COMP1_VMBADDR_MID	0xa2	8	<u> </u>
SU_WACDR_COMP1_VMBADDR_LSS	0xa3	8	!
BU_WADDR_COMP2_VMBADDR_MSB	0xa5	2	Test only
BU_WADDR_COMP2_VMBADDR_MID	0xa6	8	<u> </u>
BU_WADDR_COMP2_VMBADDR_LSB	0x27	9	!



Kaybola Register Name	Keyhote	3.4	6
Keyhole Register Name	- Address	ಶೀರ	Comments
BU_WADCR_VBACDR_MSB	0xa9	2	Test only
SU_WADOR_VBADOR_MID	0xaa	8	į
BU_WADDR_VBADDR_LSB	0xao	8	<u>.</u>
BU_WADDR_COMPO_HALF_WIDTH_IN_BLOCKS_MSB	0xad	2	Must be
BU_WADCR_COMPO_HALF_WIDTH_IN_BLOCKS_MID	0xae	8	Loaded
BU_WADDR_COMPO_HALF_WIDTH_IN_BLOCKS_LSB	0xal	8	<u>.</u>
BU_WADDR_COMP1_HALF_WIDTH_IN_BLOCKS_MSB	0xb1	2	Must be
BU_WADDR_COMP1_HALF_WIDTH_IN_BLOCKS_MID	0xb2	8	Loaded
BU_WADDR_COMP1_HALF_WIDTH_IN_BLOCKS_LSB	0xb3	9	j
BU_WACOR_COMP2_HALF_WIDTH_IN_BLOCKS_MSB	0xb5	2	Must be
BU_WADDR_COMP2_HALF_WIDTH_IN_BLOCKS_MID	0xb6	a	Lcaded
BU_WADDR_COMP2_HALF_WIDTH_IN_BLOCKS_LSB	0xb7	8	
BU_WADDR_HB_MSB	0xb9	2	Test only
BU_WADOR_HB_MID	0xba	8	
BU_WAOOR_HB_LSB	0xbb	8	
BU_WADDR_COMPO_OFFSET_MSB	0xbd	2	Must be
BU_WADDR_COMPO_OFFSET_MID	Oxbe	8	Loaded
BU_WADDR_COMPO_OFFSET_LSB	Oxbf	8	
BU_WADDR_COMP1_OFFSET_MSB	0xc1	2	Must be
BU_WADDR_COMP1_OFFSET_MID	0xc2	8	Loaded
BU_WADDR_COMP1_OFFSET_LSB	0xc3	8	
BU_WADDR_COMP2_OFFSET_MSB	0xc5	2	Musi be
BU_WADDR_COMP2_OFFSET_MID	0xc6	8	Loaded
BU_WADDR_COMP2_OFFSET_LSB	0xc7	8	
"BU_WADDR_SCRATCH_MSB	0xc9	2	Test only
BU_WADDR_SCRATCH_MID	0xca	8	
BU_WADDR_SCRATCH_LSB	0xcb	8	
BU_WADDR_MBS_WIDE_MSB	0xcd	2	Must be
BU_WADDR_MBS_WIDE_MID	0xce	8	Loaded
BU_WACOR_MBS_WIDE_LSB	0xcl	8	
BU_WADDR_MBS_HIGH_MSB	0xd1	2	Must be
BU_WADDR_MBS_HIGH_MID	0xd2	8	Loaced
BU_WACOR_MBS_HIGH_LSB	0xd3	8	



Keyhole Register Name	Keynole Address	Bits	Comments
BU_WACCR_CCMP0_LAST_MB_IN_ROW_MSB	0xd5	2	Must be
BU_WACCR_COMPO_LAST_MB_IN_ROW_MID	0xd6	8	Loaded
BU_WACOR_COMPO_LAST_MB_IN_ROW_LSB	0xd7	8	<u> </u>
BU_WACOR_COMPI_LAST_MB_:N_ROW_MSB	0xd9	2	Musi be
BU_WADDR_COMP1_LAST_MB_IN_RCW_MID	Oxda	8	Loacec
BU_WADDR_COMPI_LAST_MB_IN_ROW_LSB	0xdb	8]
BU_WACDR_CCMP2_LAST_MB_IN_ROW_MSB	0xdd	2	Musi de
BU_WADDR_COMP2_LAST_MB_IN_ROW_MID	Oxde	8	Loases
BU_WADDR_COMP2_LAST_MB_IN_ROW_LSB	0xdf	8	:
BU_WADDR_COMPO_LAST_MB_IN_HALF_ROW_MSB	0xe1	2	Must te
BU_WADDR_COMPO_LAST_MB_IN_HALF_ROW_MID	0xe2	8	Loaced
BU_WACOR_COMPO_LAST_MB_IN_HALF_ROW_LSB	0xe3	8	
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MSB	0xe5	2	Must be
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MID	0xe6	8	Loaded
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_LSB	0xe7	8	
BU_WAGGR_COMP2_LAST_MB_IN_HALF_ROW_MSB	0xe9	2	Musi be
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_MID	0xea	8	Loaded
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_LSB	0xeb	8	
BU_WACDR_COMPO_LAST_ROW_IN_MB_MSB	0xed	2	Must be
BU_WADDR_COMPO_LAST_ROW_IN_MB_MID	Oxee	8	Loaded
BU_WADDR_COMPO_LAST_ROW_IN_MB_LSB	Oxel	8	!
BU_WADDR_COMP1_LAST_ROW_IN_MB_MSB	0xf1	2	Must be
BU_WADDR_COMP1_LAST_ROW_IN_MB_MID	0xf2	8	Loaded
BU_WADDR_COMP1_LAST_ROW_IN_MB_LSB	0x13	8	
BU_WADDR_COMP2_LAST_ROW_IN_MB_MSB	0x15	2	Must be
BU_WADDR_COMP2_LAST_ROW_IN_MB_MID	0xf6	8	Loaded
BU_WADDR_COMP2_LAST_ROW_IN_MB_LSB	0x17	8	
BU_WADDR_COMPO_BLOCKS_PER_MB_ROW_MSB	0x 19	2	Must be
BU_WACDR_COMPO_BLOCKS_PER_MB_ROW_MID	Oxfa	8	Loaded
BU_WADDR_COMPO_BLOCKS_PER_MB_ROW_LSB	0xfb	8	
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MSB	0x1d	2	Must be
SU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MID	0xfe	8	Loades
BU_WACCR_COMPI_BLOCKS_PER_MB_ROW_LSB	0xff	8	



		-,	
Keyhole Register Name	Keynole		
	Accress	3,3	Comments
BU_WACCR_COMP2_BLOCKS_PER_MB_ROW_MSB	0x101	2	Musice
BU_WACCF_COMP2_BLOCKS_PER_MB_ROW_MID	0x102	8	Loaded
BU_WACCR_COMP2_BLOCKS_PER_MB_ROW_LSB	0x103	8	-
BU_WACOR_COMPO_LAST_MB_ROW_MSB	0x105	2	Mustice
BU_WADDR_COMPO_LAST_MB_ROW_MID	0x106	8	Loaded
; BU_WADDR_COMPO_LAST_MB_ROW_LSB	0x107	8	Ī
BU_WADDR_COMP1_LAST_MB_ROW_MSB	0x109	2	Mus: te
BU_WACOR_COMP1_LAST_MB_ROW_MID	Cx10a	3	Loaced
BU_WADER_COMP1_LAST_MB_ROW_LSB	0x10b	8	
BU_WACCR_COMP2_LAST_MB_ROW_MSB	0x10d	2	Must be
BU_WADDR_CCMP2_LAST_MB_ROW_MID	0x10e	8	Loaded
BU_WADDR_COMP2_LAST_MB_ROW_LSB	0x10f	8	
BU_WADER_COMPO_HBS_MSB	0x111	2	Must be
BU_WACCR_COMPO_HBS_MID	0x112	а	Loaded
BU_WACDR_COMPO_HBS_LSB	0x113	8	
BU_WACOR_COMP1_HBS_MSB	0x115	2	Must be
BU_WADDR_COMP1_HBS_MID	0x116	8	Loaded
BU_WACDR_COMP1_HBS_LSB	0x117	8	
BU_WACOR_COMP2_HBS_MSB	0x119	2	Must be
BU_WADDR_COMP2_HBS_MID	Oxila	a !	Loaded
BU_WADDR_COMP2_HBS_LSB	0x11b	8 ;	
BU_WADDR_COMPO_MAXHB	0x11f	2	Musi be
BU_WADDR_COMP1_MAXHB	0x123	2	Loaded
BU_WADDR_COMP2_MAXHB	0x127	2	•
BÜ_WACOR_COMPO_MAXVB	Cx12b	2	Must be
BU_WADDR_COMP1_MAXVB	0x12!	2	Loaded
BU_WADDR_COMP2_MAXVB	Cx133	2	i !

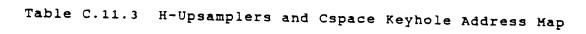
C.12.3 Horizontal Upsampler and Color Space Converter Keyhole.

. . .



Table C.11.3 H-Upsamplers and Cspace Keyhole Address Map

Keyhole Pegister	Keyhole		
Name	Address	Bits	Comment
BU_UHO_ACO_1	0x0	5	F/W- Coeff 0,0
SU_UHC_A00_0	Oxi	8	
BU_UH0_A01_1	0x2	5	R/W- Coeff 0,1
BU_UH0_A01_0	0x3	8	
BU_UH0_A02_1	0x4	5	PvW-Coeff 0,2
BU_UH0_A02_0	0x5	8	
BU_UH0_A03_1	0x5	5	F/W- Coeff 0.0
BU_UH0_A03_0	0x7	8	
BU_UH0_A10_1	0x8	5	FVW- Coeff 1,0
BU_UH0_A10_0	0 x9	8	
BU_UH0_A11_1	0xa	5	PW-Coeff 1,1
BU_UH0_A11_0	0xb	8	
BU_UH0_A12_1	0xc	5	P/W- Coeff 1,2
BU_UH0_A12_0	0xd	8	
BU_UH0_A13_1	0xe	5	P/W- Coeff 1,3
BU_UH0_A13_0	0xf	8	
BU_UH0_A20_1	0x10	5	RW- Coeff 2,0
BU_UH0_A20_0	0x11	8	
BU_UH0_A21_1	0x12	5	P/W- Coeff 2,1
BU_UH0_A21_0	0x13	8	
BU_UH0_A22_1	0x14	5	R/W- Coeff 2,2
BU_UH0_A22_0	0x15	8	
BU_UH0_A23_1	0x15	5	RW- Coeff 2.3
BU_UH0_A23_0	0x17	8	
BU_UHO_MODE	0x18	2	RW
BU_UH1_A00_1	0×20	5	RVW- Coeff 0,0
BU_UH1_A00_0	0x21	8	
BU_UH1_A01_1	0x22	5	PVW- Coeff 0,1
BU_UH1_A01_0	0x23	8	
BU_UH1_A02_1	0x24	5	P/W- Coeff 0,2
BU_UH1_A02_0	C×25	8	
BU_UH1_A03_1	0×26	5	P/W- Coeff 0.0
BU_UH1_A03_0	0×27	8	



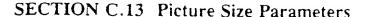
Keynore Register	Keyhole		_
`~arre	Address	3:5	Comment
BU_UH1_A10_1	0×28	5	FVW- Coeff 1.0
BU_UH1_A10_0	0±29	8	
BU_UH1_A11_1	0x2a	5	PvW- Coeff 1,1
BU_UH1_A11_0	0×25	8	
BU_UH1_A12_1	0x2c	5	RVV- Coeff 1,2
BU_UH:_A12_0	ರಿ×2ರ	8	
SU_UH1_A13_1	0x2e	5	FVW- Coeff 1,3
BU_UH1_A13_0	0x21	8	
BU_UH1_A20_1	0x30	5	RVW- Coeff 2.0
3U_UH!_A20_0	0x31	8	
8U_UH1_A21_1	0x32	5	RVV- Coeff 2.1
BU_UH1_A21_0	0x33	8	
BU_UH1_A22_1	0x34	5	FVW- Coeff 2.2
BU_UH1_A22_0	0x35	8	
BU_UH1_A23_1	0x36	5	R/W- Coeff 2.3
BU_UH1_A23_0	0x37	8	
BU_UH1_MODE	0x38	2	P/W
BU_UH2_A00_1	0x40	5	R/W- Coeff 0,3
BU_UH2_A00_0	0x41	8	
BU_UH2_A01_1	0x42	5	PW- Coeff 0,1
BU_UH2_A01_0	0x43	8	
BU_UH2_A02_1	0x44	5	FVW- Coeff 0.2
BU_UH2_A02_0	0x45	8	
BU_UH2_A03_1	0x46	5	R/W- Coeff 0,0
BU_UH2_A03_0	0x47	8	<u> </u>
BU_UH2_A10_1	0x48	5	R/W- Coeff 1,3
BU_UH2_A10_0	0x49	8	
BU_UH2_A11_1	Cx4a	5	PVW- Coeff 1,1
8U_UH2_A11_0	0x4b	8	
BU_UH2_A12_1	0x4c	5	RW- Coeff 1.2
BU_UH2_A12_0	0x4d	8	
SU_UH2_A13_1	Cx4e	5	R/W- Coeff 1.3
BU_UH2_A13_0	Cx4f	8	



Table C.11.3 H-Upsamplers and Cspace Keyhole Address Map

Keynole Register	Keyhole		
Name	Address	3its	Comment
BU_UH2_A20_1	C×50	5	PVV- Coeff 2.0
3U_UH2_A20_0	0x51	8	ļ
5U_UH2_A21_1	0x52	5	FVW- Coeff 2.1
BU_UH2_A21_0	0x53	а	
BU_UH2_A22_1	Cx54	5	RW- Coeff 2.2
BU_UH2_A22_0	0x55	в	
BU_UH2_A23_1	0x56	5	P/W- Coeff 2.3
BU_UH2_A23_0	0x57	8	
BU_UH2_MODE	0×58	2	P/W
BU_CS_A00_1	0x60	5	₽₩
BU_CS_ACO_0	0x61	8	
3U_CS_A10_1	Cx62	5	P-VV
8U_CS_A10_0	0x63	8	
BU_CS_A20_1	0x64	5	P/W
8U_CS_A20_0	0x65	8	
BU_CS_80_1	0x66	6	PAY
BU_CS_B0_0	0x67	8	
BU_CS_A01_1	0x68	5	RW
BU_CS_A01_0	0x69	8	
BU_CS_A11_1	0x6a	5	₽₩
BU_CS_A11_0	0x6b	8	
BU_CS_A21_1	0x6c	5	₽₩
BU_CS_A21_0	0x6d	8	
8U_CS_81_1	0x6e	6	RW
8U_CS_81_0	0x6f	8	
BU_CS_A02_1	-0x70	5	Pw
BU_CS_A02_0	0x71	8	
BU_CS_A12_1	0x72	5	P/W
BU_CS_A12_0	0x73	8	
3U_CS_A22_1	0x74	5	₽w į
BU_CS_A22_0	0x75	8 I	
BU_CS_B2_1	0x76	6	P/W
BU_CS_B2_0	0x77	8	

10



C.13.1 Introduction

The following stylized code fragments illustrate the processing necessary to respond to picture size interrupts from the write address generator. Note that the picture size parameters can be changed "on-the-fly" by sending combinations of HORIZONTAL MBS, VERTICAL MBS, DEFINE SAMPLING (for each component) tokens, resulting in write address generator interrupts. These tokens may arrive in any order and, in general, any one should necessitate the re-calculation of all of the picture size parameters. At setup time, however, it would be more efficient to detect the arrival of all of the events before performing any calculations.

It is possible to write specific values into the picture size parameter registers at setup and, therefore, to not rely on interrupt processing in response to tokens. For this reason, the appropriate register values for SIF pictures are also given.

20 C.13.2 Interrupt Processing for Picture Size Parameters

There are five picture size events, and the primary response of each is given below:

```
if (hmbs_event)
  load(mbs_wide).
else if (vmbs_event)
  load(mbs_high).
else if (def_samp0_event)
{
  load (maxhb(0)).
  load (maxvb(0)):
}
else if (def_samp1_event)
{
  load (maxvb(1)):
  load (maxvb(1)):
}
else if (def_samp1_event)
{
  load (maxvb(1)):
}
load (maxvb(1)):
}
```

In addition, the following calculations are necessary to retain consistent picture size parameters:

```
if (hmbs_event||umbs_event||
    def_samp0_event||def_samp1_event||def_samp2_event;

for (i=0; i<max_component; i++)

{
    hbs(i) = addr_hbs(i] = (maxhb(i)+1) * mbs_wide;
    naif_width_in_blocks(i) = ((maxhb(i)+1) * mbs_wide)/2;
    last_mb_in_row(i) = hbs(i) + (maxhb(i)+1);
    last_mb_in_half_row(i) = half_width_in_blocks(i) +
    maxhb(i)+1);
    last_row_in_mb(i) = hbs(i) * maxvb(i);
    blocks_per_mb_row(i) * last_row_in_mb(i) + hbs(i);
    last_mb_row(i) = blocks_per_mb_row(i) * (mbs_hign-1);
}</pre>
```



Although it is not strictly necessary to modify the dispaddr register values (such as the display window size) in response to picture size interrupts, this may be desirable depending on the application requirements.

5 C.13.3 Register Values for SIF Pictures

The values contained in all the picture size registers after the above interrupt processing for an SIF, 4:2:0 stream will be as follows:

C.13.3.1 Primary Values

BU_WADDR_MBS_WIDE = 0x16

BU_WADDR_MBS_HIGH = 0x12

BU_WADDR_COMP0_MAXHB = 0x01

BU_WADDR_COMP1_MAXHB = 0x00

BU_WADDR_COMP2_MAXHB = 0x00

BU_WADDR_COMP0_MAXVB = 0x01

BU_WADDR_COMP1_MAXVB = 0x00

BU_WADDR_COMP2_MAXVB = 0x00

10 C.13.3.2 Secondary Values - After Calculation

BU_WADDR_COMP0_HBS = 0x2C

BU_WADDR_COMP1_HBS = 0x16

BU_WADDR_COMP2_HBS = 0x2C

BU_ADDR_COMP1_HBS = 0x16

BU_ADDR_COMP1_HBS = 0x16

BU_ADDR_COMP2_HBS = 0x16

BU_WADDR_COMP0_HALF_WIDTH_IN_BLOCKS = 0x16

BU_WADDR_COMP1_HALF_WIDTH_IN_BLOCKS = 0x03

BU_WADDR_COMP2_HALF_WIDTH_IN_BLOCKS = 0x03

BU_WADDR_COMP2_HALF_WIDTH_IN_BLOCKS = 0x03

BU_WADDR_COMP1_LAST_MB_IN_ROW = 0x15

BU_WADDR_COMP2_LAST_MB_IN_ROW = 0x15

BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW = 0x14

BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW = 0x0A

BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW = 0x0A

BU_WADDR_COMP0_LAST_ROW_IN_MB = 0x2C

BU_WADDR_COMP1_LAST_ROW_IN_MB = 0x0

BU_WADDR_COMP2_LAST_ROW_IN_MB = 0x0

BU_WADDR_COMP2_LAST_ROW_IN_MB = 0x0

BU_WADDR_COMP0_BLOCKS_PER_MB_ROW = 0x58

BU_WADDR_COMP1_BLOCKS_PER_MB_ROW = 0x16

BU_WADDR_COMP2_BLOCKS_PER_MB_ROW = 0x16

BU_WADDR_COMP2_LAST_MB_ROW = 0x508

BU_WADDR_COMP1_LAST_MB_ROW = 0x176

BU_WADDR_COMP1_LAST_MB_ROW = 0x176

BU_WADDR_COMP2_LAST_MB_ROW = 0x176

Note that if these values are to be written explicitly at setup, account must be taken of the multi-byte nature of most of the locations.

Note that additional Figures, which are self explanatory to those of ordinary skill in the art, are included with this application for providing further insight into the detailed structure and operation of the environment in which the present invention is intended to function.

5

10

15

20

25

30

35

The aforedescribed pipeline system of the present invention satisfies a long existing need for an improved system having an input, an output and a plurality of processing stages between the input and the output, the plurality of processing stages being interconnected by a two-wire interface for conveyance of tokens along the pipeline, and control and/or DATA tokens in the form of universal adaptation units for interfacing with all of the processing stages in the pipeline and interacting with selected stages in the pipeline for control data and/or combined control-data functions among the processing stages, so that the processing stages in the pipeline are configuration flexibility in afforded enhanced In accordance with the invention, processing. processing stages may be configurable in response to recognition of at least one token. One of the processing stages may be a Start Code Detector which receives the input and generates and/or converts the tokens.

The present invention also relates to an improved pipeline system having a spatial decoder system for video data including a Huffman decoder, an index to data and an arithmetic logic unit, and a microcode ROM having separate stored programs for each of a plurality of different picture compression/decompression standards, such programs being selectable by a token, whereby processing for a plurality of different picture standards is facilitated. The present invention may also include tokens in the form of a PICTURE_START code token for indicating that the start of a picture will follow in the subsequent DATA token, a PICTURE END token for indicating the end of an individual picture, a FLUSH token for clearing buffers and resetting the system, and a CODING_STANDARD token for conditioning the system for processing in a selected one a plurality of picture compression/decompression The present invention also relates to an standards. improved pipeline system for decoding video data and having a Huffman decoder, an index to data (ITOD) stage,

an arithmetic logic unit (ALU), and a data buffering means immediately following the system, whereby time spread for video pictures of varying data size can be controlled. Also in accordance with the invention, a processing stage receives the input data stream, the stage including means for recognizing specified bit stream patterns, whereby the processing stage facilitates random access and error recovery. The invention may also include a means for performing a stop-after-picture operation for achieving a clear end to picture data decoding, for indicating the end of a picture, and for clearing the pipeline.

The improved pipeline system may also include a fixed size, fixed width buffer, and means for padding the buffer to pass an arbitrary number of bits through the buffer. The present invention also relates to a data stream including run length code, and an inverse modeller means active upon the data stream from a token for expanding out the run level code to a run of zero data followed by a level, whereby each token is expressed with a specified number of values. The invention also includes an inverse modeller stage, an inverse discrete cosine transform stage, and a processing stage, positioned between the inverse modeller stage and the inverse discrete cosine token table responsive to a stage, transform processing data.

In addition, the present invention relates to an improved pipeline system having a Huffman decoder for decoding data words encoded according to the Huffman coding provisions of either H.261, JPEG or MPEG standards, the data words including an identifier that identifies the Huffman code standard under which the data words were coded, means for receiving the Huffman coded data words, means for reading the identifier to determine which standard governed the Huffman coding of the received data words, if necessary, in response to reading the identifier that identifies the Huffman coded data words as H.261 or MPEG Huffman coded, means operably connected to the

35

5

10

15

20

25

30

5

10

15

20

25

30

Huffman coded data words receiving means for generating an index number associated with each JPEG Huffman coded data word received from the Huffman coded data words receiving means, means for operating a lookup table containing a Huffman code table having the format used under the JPEG standard to transmit JPEG Huffman table information, including an input for receiving an index number from the index number generating means, and including an output that is a decoded data word corresponding to the index number.

The improved system includes a multi-standard video decompression apparatus having a plurality of stages interconnected by a two-wire interface arranged as a Control tokens and DATA pipeline processing machine. Tokens pass over the single two-wire interface for carrying both control and data in token format. decode circuit is positioned in certain of the stages for recognizing certain of the tokens as control tokens pertinent to that stage and for passing unrecognized control tokens along the pipeline. Reconfiguration processing circuits are positioned in selected stages and responsive to a recognized control reconfiguring such stage to handle an identified DATA A wide variety of unique supporting subsystem circuitry and processing techniques are disclosed for implementing the system.

It will be apparent from the foregoing that, while particular forms of the invention have been illustrated and described, various modification can be made without departing from the spirit and scope of the invention. Accordingly, it is not intended that the invention be limited, except as by the appended claims.